THE PHOTO-REALISTIC AFIT VIRTUAL COCKPIT

THESIS

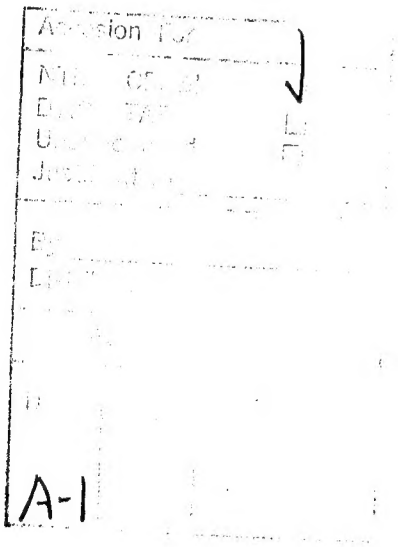
Milton Eli Diaz, BS
Captain, USAF

AFIT/GCS/ENG/94D-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-02



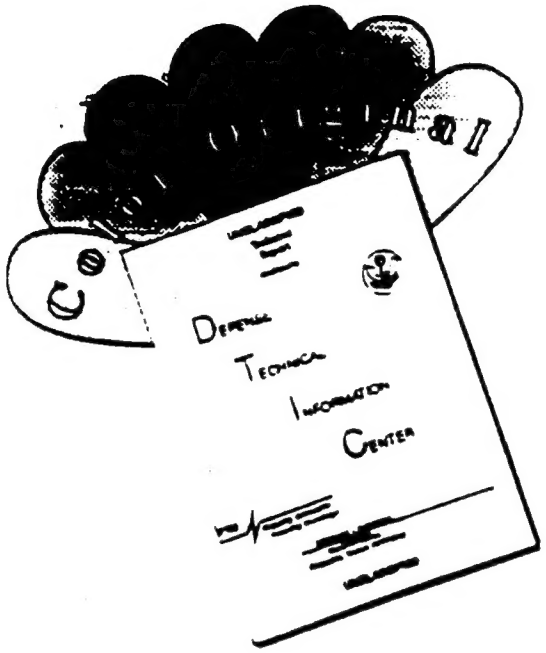
THE PHOTO-REALISTIC AFIT VIRTUAL COCKPIT

THESIS

Milton Eli Diaz, BS
Captain, USAF

AFIT/GCS/ENG/94D-02

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

AFIT/GCS/ENG/94D-02

THE PHOTO-REALISTIC AFIT VIRTUAL COCKPIT

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

Milton Eli Diaz, BS

Captain, USAF

December, 1994

Acknowledgments

I would like to thank all those whose support made my AFIT experience enjoyable. Special thanks go to Col. Stytz, my thesis advisor, Dr. Lamont, the graphics gang: John Vanderburgh, Jim Kestermann, and J.J. Rohrer, my buddies Karl Pfeiffer and Bill Wood, and most of all, my wife, Florence.

Table of Contents

List of Figures

List of Table

Abstract

The Air Force Institute of Technology (AFIT) has pursued research in virtual environments since 1988. This research expands the current capabilities of the AFIT Virtual Cockpit (VC) by increasing the realism of the cockpit environment and improving the pilot's command interface. Realism is improved creating console elements from texture maps and polygonal models; these elements include working dials, switches and circuit breakers. The pilot command interface is improved in part by adapting the AFIT Information Pod using a two-dimensional mouse input to the virtual three-dimensional environment. This immersive virtual environment is also improved by modifications to the Head Mounted Display (HMD) reducing jitter and allowing the simulation pilot to adjust his/her position within the cockpit.

The Photo-Realistic AFIT Virtual Cockpit

I. Introduction

1.1 Overview

The Air Force Institute of Technology (AFIT) Virtual Cockpit (VC) is a virtual reality cockpit simulation project that examines graphics and human factors issues in a fully immersive environment. The primary goal of the VC is to develop the technology for a low cost, distributed interactive simulation (DIS) compatible flight simulator that can be used for large scale tactical training. Currently, the cost and availability of fully functional dome flight simulators are prohibitive for large scale interactive simulations. Chapter I discusses the importance of combat training and flight simulators and the rationale for photo-realistic virtual environment flight simulations. Also, this chapter expounds on the problem statement, scope, methodology, and constraints of the research.

1.2 Background

According to Dyer, soldiers learn more about combat in the first thirty seconds of battle than they learn in all the training beforehand. "Combat is the ultimate reality that Marines - or any other soldiers, under any flag - have to deal with" (Dyer). However, soldiers must survive their

first exposure to war before they may learn from their combat experience. Although military training can not fully prepare people for combat, quality training can provide soldiers with the knowledge needed for survival and eventual victory. On the importance of training, Clausewitz states: (Clausewitz)

> "It is immensely important that no soldier, whatever his rank, should wait for war to expose him to those aspects of active service that amaze and confuse him when he first comes across them."

The Value of Training. History provides countless instances of how a nation's military has benefited from training or suffered from lack of training. For example, at the turn of the Nineteenth century, Lord Nelson's English fleet defeated two larger and better armed French fleets at the battles of the Nile and Trafalgar. Nelson's chief advantage in both battles can be attributed to his highly qualified captains and well trained crews. The English Navy had the luxury of combat drill and maneuver at sea while the French had to train in port under English blockade (Taylor). Recently, the Gulf War, fought between Iraq and a coalition of several nations led by the United States military under the leadership of General Schwarzkopf, provided one more example of how a military benefits from superior training. Because Iraq's army was considered a veteran force and was regarded as the world's fourth largest, General Schwarzkopf refused to fill out his force with the National Guard's "roundout brigades" because he believed the units would not have received adequate training before the Desert Storm offensive. Instead, General Schwarzkopf employed his well trained army boldly to drive the enemy from the field (Schwarzkopf).

The best training recreates battle field conditions at as many levels as possible (i.e. from the tactical decisions a pilot executes during a dog fight to the strategic decisions made by the commanding general). Peacetime maneuvers such as Red Flag and REFORGER, a combined arms exercise held in Central Europe, provide combat, support, and command units with insight into the confused coordination of a battle. However, moving troops to and from maneuvers is expensive and, as a result, only a few units may participate at any given time. Simulations provide a cost effective supplement to actual field maneuvers (Gerhard; Rolfe and Staples).

Flight Simulators. Flight simulators have been vastly used since their inception over

eighty years ago. Over the years, there have been considerable improvements from the first simulators, typically an actual aircraft on a universal joint, to the present day flight simulators. Simulators have progressed from propeller to jet aircraft simulators and from mechanical controlled to fly-by-wire (Markman). By 1980, there were more than 300 modern dome flight simulators that projected a computer generated image onto a dome screen (Rolfe and Staples). These modern simulators consist of a: visual system, motion base, cockpit, and dynamic aircraft model (Markman). The simulator's capabilities are based on the simulation's purpose. For instance, because a cockpit procedures trainer provides training for basic air crew duties, it does not need to be as complex as a modern flight simulator which attempts to recreate realistic flight sensations such as motion and acceleration (Hazer and Ringler).

Flight simulators are used either as tools for aircraft development or pilot training. Among recently used military simulators are the Large Amplitude Multimode Aerospace Research Simulator (LAMARS), shown in Figure 1-1, located at the Flight Dynamics Laboratory at Wright-Patterson AFB and the Naval Air Test Center's F/A-18 simulator used to accurately study the operation of all the F/A-18's systems (Markman).

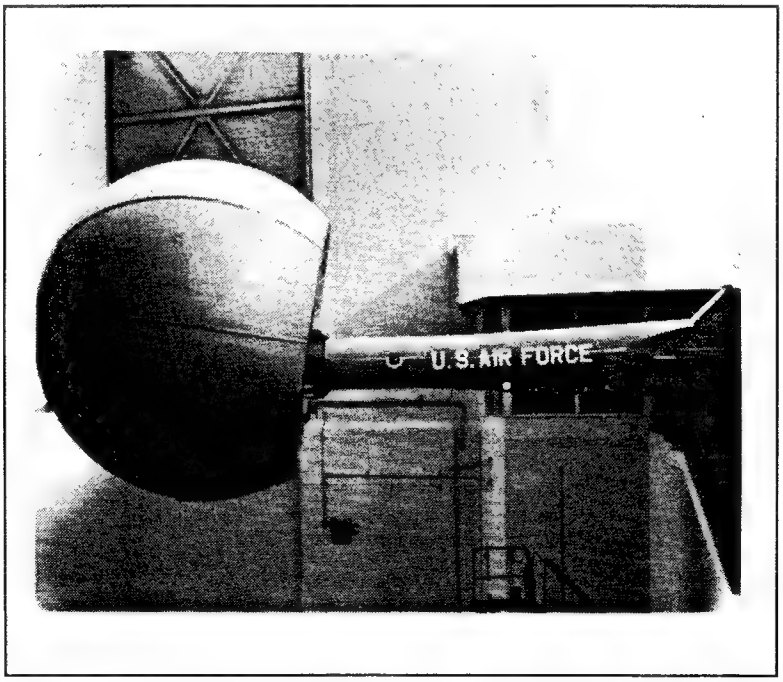


Figure 1-1. The LAMARS Full Fidelity Simulator at Wright-Patterson AFB (Markman).

According to Markman, in-flight simulators are aircraft that possess feel and flying characteristics that can be modified to simulate another aircraft. In-flight simulators have been used for various aircraft development programs, such as the F-16, YF-17, F-18, B-1, and the space shuttle, and for highly specialized training. The YF-17's potential for pilot induced oscillation was discovered and corrected due to in-flight simulation before the YF-17's first flight. The USAF/USN/Calspan Learjet is used for training Air Force and Navy test pilots. (Markman)

1.3 Rationale

One method the Air Force uses to alleviate the cost of training (i.e. missile combat crew training, pilot training) is the interactive simulator. For a command post exercise, the actual command post area serves as the interactive simulator. The participants react to information received from the exercise coordinators on UHF radio and telephone lines. The coordinators present realistic scenarios that emphasize command post procedures, such as coordinating security and disaster responses. Because information flows into and out of the command post through normal channels, the simulation provides all the participants with a realistic experience.

Another method of simulation used in the Air Force is the full scale mock-up. In contrast to a command post exercise, a mock-up attempts to present a safe environment where training can address challenging or dangerous concepts which are impractical to address in the field. For example, the Missile Procedures Trainer (MPT), a life size model of a Missile Control Capsule, is used to train combat crews in emergency procedures such as injury evacuation of personnel and special fire fighting techniques. Like the command post exercise, the MPT training session presents combat crews with information channeled through a variety of input modes such as telephone lines, audio alarms, warning lights, and computer printouts. Typically, the two MPTs in a Missile Group are active eighteen hours a day each and five to six days a week in order to complete the Group's monthly training requirements (Earle).

A flight simulator is a much larger and more expensive mock-up than an MPT. For the highest quality simulation, the flight simulator requires a large dome screen to present the

out-the-canopy view and lifts to provide the necessary visual and motion cues. The displays and lifts used in a simulator are two of the factors that impact the cost of flight simulators.

In addition to size and display, a number of factors influence the final cost of a standard simulator. The Ross-Yarger model for simulator cost analysis lists the following factors: (Ross and Yarger)

1. computer processing power
2. number of physical crew stations
3. degrees of freedom
4. number of flight & aircraft sensations presented
5. simulator's weight
6. rate of power consumption
7. number of emergency procedures simulated
8. cooling required to run the simulator

Cost factors increase dramatically as the aircraft being simulated becomes more complex and as a result, an operational simulator may cost in excess of forty million dollars (Stytz; Rolfe and Staples). The expense and space required for the traditional simulator limits its availability for training pilots.

As actual flight hours grow less accessible because of the cost incurred or unavailability of an aircraft, the need for more cost effective and readily available simulators has become clear. Advances in virtual environment technology have created realistic flight simulators that take a fraction of the space old simulators required and cost a hundredth of the price of a dome simulator. The cost savings stems from using relatively low cost, high resolution displays controlled by a computer graphics workstation (Switzer).

1.4 Problem Statement

AFIT has pursued research in virtual environments since 1988. In particular, the Advance Research Project Agency (ARPA) has sponsored the AFIT VC that has investigated the feasibility of a low cost, combined arms force, distributed network cockpit simulator. This thesis addresses two problems: 1) to increase the realism of the VC and 2) expand the pilot's potential command interface in the virtual environment. The first problem entails developing a photo-realistic cockpit

with working and relatively functional dials, switches, and circuit breakers. The second problem requires adapting techniques used to input commands into a VR environment with the photo-realistic controls within the VC.

1.5 Scope

This research addresses the following objectives: 1) realistically depict the F-15E forward crew station (Figure 1-2) using polygonal objects and texture maps, 2) demonstrate a high quality pilot immersive interface which allows the pilot to interact with the instrumentation, and 3) accomplish the former while maintaining at least a 15 Hz frame rate - the minimum rate needed for a smooth interactive simulation.

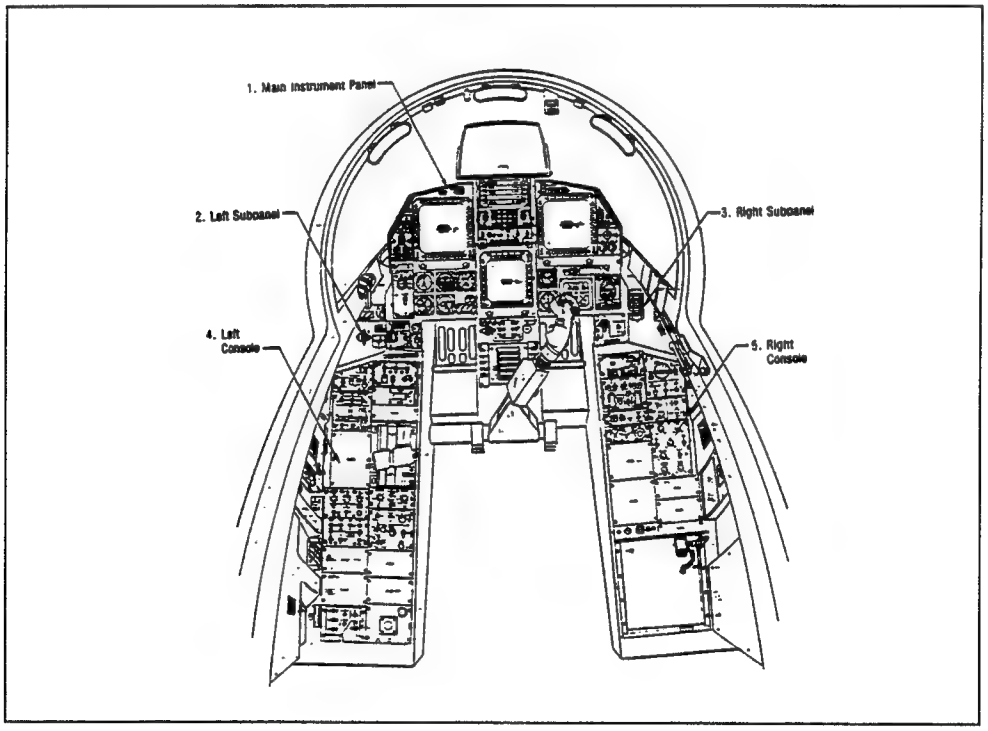


Figure 1-2. F-15E Forward Crew Station (Aeronautical).

Realistic Depiction. In order for the VC to provide an effective training environment, the VC simulation must create a sense of reality. Two factors contributing to the realistic feel of the simulation are the accuracy of the objects depicted and the frame rate of the simulation. Accuracy refers to the amount of detail represented in the object. The more detail present, the more realistic the rendered object appears. Equally important, frame rate is the number of scenes displayed on screen per second. A high frame rate is desirable because it results in a smooth simulation. However, these two factors can work against each other in simulations. For a modern graphics system to render motion in a believable fashion, it must fully present all the polygons of every object on the screen before the next frame is depicted. Otherwise, the resulting simulation appears sluggish and hesitant as seen in a slow motion film. As simulations become more realistic, the complexity of the objects in the scene can quickly overwhelm the graphical display equipment (DeHaemer and others).

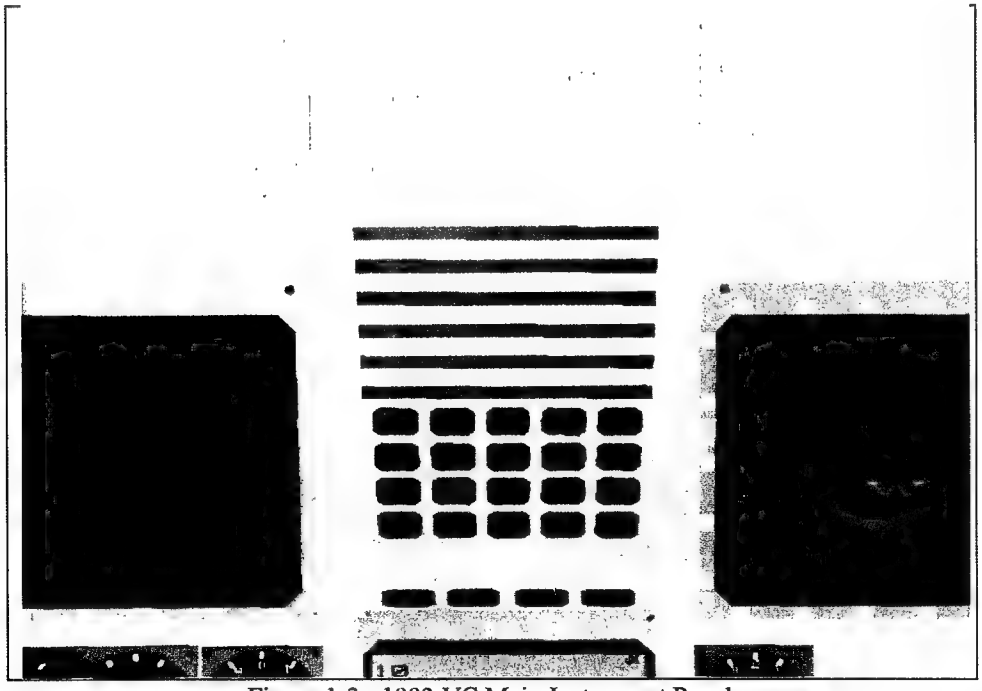


Figure 1-3. 1993 VC Main Instrument Panel.

Depicting a photo-realistic VC requires a combination of polygonal objects and texture mapping techniques. The polygonal objects will form the VC's instruments, consoles, switches, and knobs and will be modeled and edited to reduce the number of polygons used. Individually, simple models can not capture the realism desired. Although the polygonal objects can be moved and are three dimensional constructs, they can appear flat and cartoonish as Figure 1-3 shows. On the other hand, texture maps can capture a photograph's realism, but can not be manipulated. Figure 1-4, the photograph of an F-15E front instrument panel could be used as a texture map (DeHaemer and others; McCarty). Figure 1-4 also displays many moving switches and dials which need to be manipulated to provide realism. The VC will use the complementary features of texture maps and polygonal objects to create a photo-realistic environment.

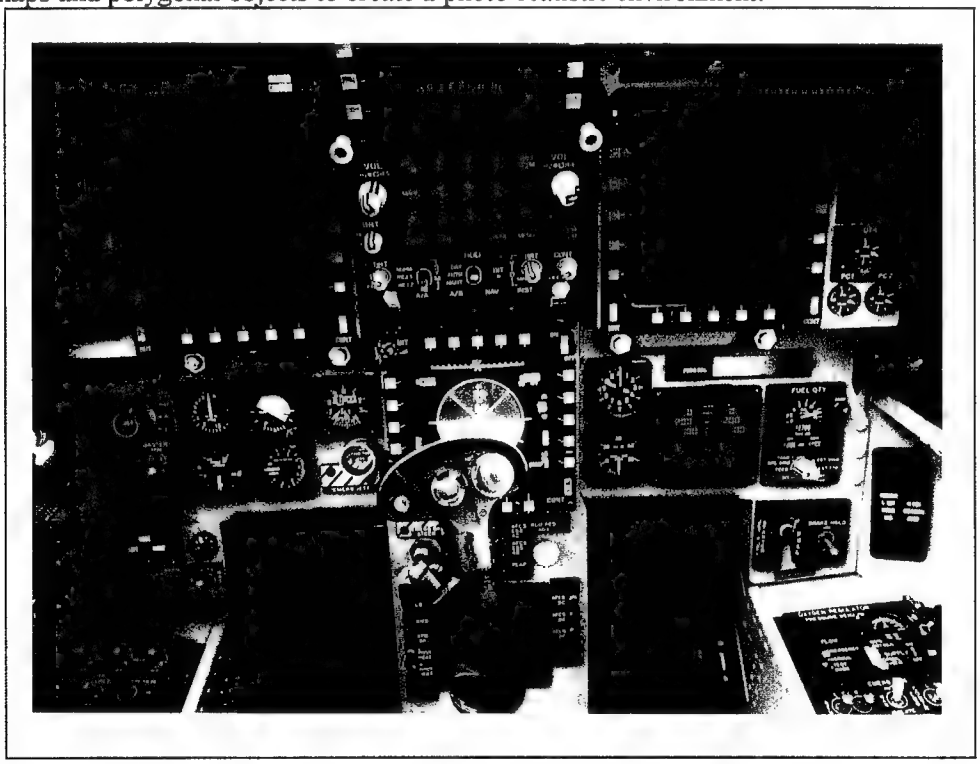


Figure 1-4. Photograph of F-15E Simulator Main Instrument Panel .

Immersive Interface. Providing an intuitive VC interface will require a conceptually simple and easy to learn method for the pilot to interact with his surroundings. The primary method examined in this paper is a head mounted display (HMD) used in combination with a

mouse interface based on the AFIT Information Pod developed by Kestermann. The mouse interface has several advantages over other input interfaces which make the mouse a practical selection. First, the mouse is easy to use. The mouse's point-and-click are easily mastered by most people familiar with home computers. Second, the mouse does not require very much memorization. For example, for a keyboard interface, the user must memorize the sequence of key strokes needed to accomplish an action. Also, the mouse is a more mature technology than the three-dimensional interfaces available. Compared to the mouse, three dimensional devices such as the spaceball and DataGlove exhibit control and/or jitter problems that the mouse does not have. Finally, the mouse is readily available and does not require special setup or calibration unlike the DataGlove or the spaceball (Kestermann).

Frame Rate. The frame rate is directly related to the level of detail, i.e. the number of polygonal objects needed to depict an image. The more polygons needed in the model, the slower the frame rate will be. Because texture maps use fast memory operations rather than complex computations, complex looking objects are rendered relatively quickly (McCarty). However, even for powerful workstations, complex simulations can stress the machine's abilities. The AFIT Graphics Laboratory's Silicon Graphics Iris 4D workstations can render texture maps in real-time. However, the frame rate of inherently complex interactive simulations visibly slows when detailed texture maps are used. Because the VC's frame rate usually runs between 13 Hz and 16 Hz (Erichsen), the VC must be carefully adapted to achieve the photo-realistic level of detail without losing the frame rate continuity during the simulation.

1.6 Performer and ObjectSim

Performer. IRIS Performer is an application development environment designed to interface efficiently with the SGI graphics rendering hardware. Performer's main components are two ANSI C object code libraries and their corresponding header files. The first library (*libpr*) is a low level library which optimizes rendering functions, state controls, and other fundamental IRIX operating system and IRIS Graphics library functions needed for real-time graphics. Because *libpr*

contains machine-specific elements to support particular workstation models, visual applications using Performer libraries are portable across Silicon Graphics platforms. The other library (*libpf*) is the visual simulation development environment that facilitates the multiprocessing, database traversal and rendering systems by efficient use of *libpr* library functions. Although *libpf* is layered on top of *libpr*, both libraries are easily accessible to a simulation application. Additionally, when using Performer, a visual application also has access to the IRIX operating system and IRIS Graphics Library on which Performer is layered (McLendon).

ObjectSim. ObjectSim uses the Performer interface to split the application, draw, and cull operations of the VC simultaneously across the SGI processors. As a result, the VC processes are synchronized so that no single process outpaces another and the simulation's frame rate runs smoothly. ObjectSim serves as the connection between the simulation and the outside network. To allow the visualization of the network traffic, ObjectSim correlates information from objects broadcasting over the network with the models available on the SGIs. An object that broadcasts information over the network is called a "player" in the ObjectSim application.

1.7 Virtual Cockpit

The 1993 AFIT VC is a very capable virtual flight simulator. The VC has the following features : 1) sensor suite, 2) weapons systems, 3) DIS compatibility, and 4) realistic flight dynamics (Erichsen; Gerhard; Sheasby). These features provide the fundamental set of tools required for the VC to participate in a combined force network simulation. In fact, the 1993 VC successfully participated in ARPA-sponsored WARBREAKER exercises (Erichsen). With relatively minor changes such as updating engine, flight dynamics, and fuel flow routines, the VC can take part as a player in a realistic network simulation. However, to provide a valuable pilot training environment, the 1993 VC still requires additional work which includes the photo-realistic displays and controls resulting from this thesis.

1.8 Methodology

The original object-oriented approach to the AFIT VC was established for ease of maintenance and modification (Rumbaugh and others). Each year since the research started in 1988, the object-oriented design has facilitated expanding the VC's capabilities. This thesis will continue with the incremental approach to modifying the VC.

The photo-realistic cockpit requires completion of two sub-goals: 1) generating the photograph quality image and 2) ensuring the dials, switches, and circuit breakers behave correctly. An incremental process is used to address the previously stated sub-goals. Each functional component such as the landing gear panel of the cockpit is modeled before beginning work on the next component. Finally, the simulation is modified when the models are completed.

Model Creation. The first imaging steps are creating a scale model of the control panels followed by mapping the location of each dynamic element on the panels. To complete the image, texture maps of essential non-moving parts of the panels, such as dial faces and labels, are generated. Finally, the panels are completed by applying the moving elements to their corresponding locations. MultiGen Flight, a version of MultiGen that supports Software Systems' Flight database format, is used to create the photo-realistic static and dynamic models (Software).

MultiGen, an interactive tool for creating and editing visual databases, has been implemented in a number of visual systems. Each implementation contains a Database Logic (DBL), a common user interface plus the software subsystems needed to support a specific visual platform. A MultiGen database is a file describing the visual scene. Usually, a single database file represents the terrain and static models, and each moving object is placed in a separate file. The VC, however, is a complex construction which includes multiple files as children.

In the VC model's hierarchy, the static and dynamic parts of the F-15E are the children of the airframe (Figure 1-5). Because the airframe is a dynamic model moving through the virtual environment and containing moving and non-moving parts, its parts are, in effect, dynamic in relation to the virtual environment. The VC's structure has a modular hierarchy - containing multiple static and dynamic files - in line with the modular coding techniques that will be used.

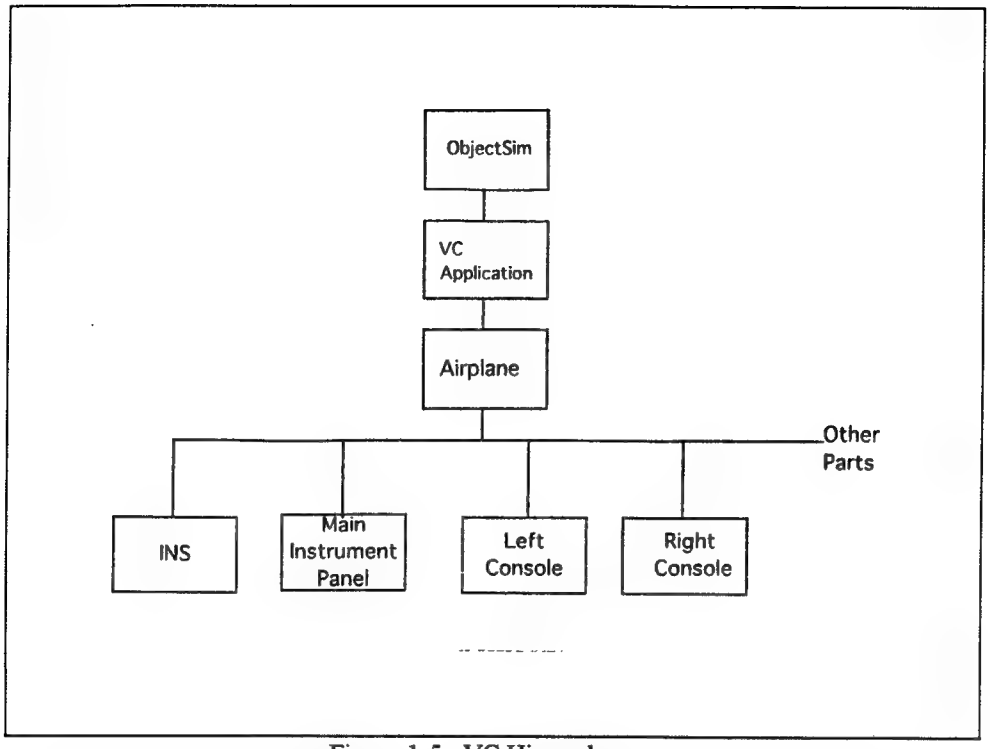


Figure 1-5. VC Hierarchy.
The parent node, the airframe, has multiple static and dynamic children. The main instrument panel and the left and right consoles are static models and are children of the airframe.

Model Reaction. After creating the images, switches, and circuit breakers are incorporated into the code as dynamic components. The first step in this portion requires modifying an existing panel, such as the landing gear panel visible in Figure 1-2, to react to the user's "touch". To accomplish this, the mouse input method developed in the AFIT Information Pod (Kestermann) is adapted for transmitting the pilot's touch.

One of the resulting actions from any pilot input is to 'move' the switch the pilot has pushed. Once the move has been shown, any number of resulting actions can also be depicted. For example, if the landing gear knob (Figure 1-6) is lowered, the resulting actions would include lowering a landing gear model and adding more drag to the aircraft's drag coefficient.

At each point in the research process where the new elements are fused to the VC, the system's code will be recompiled and tested. The object-oriented coding practices in place will simplify measurement of the VC's frame rate, the main barometer for success.

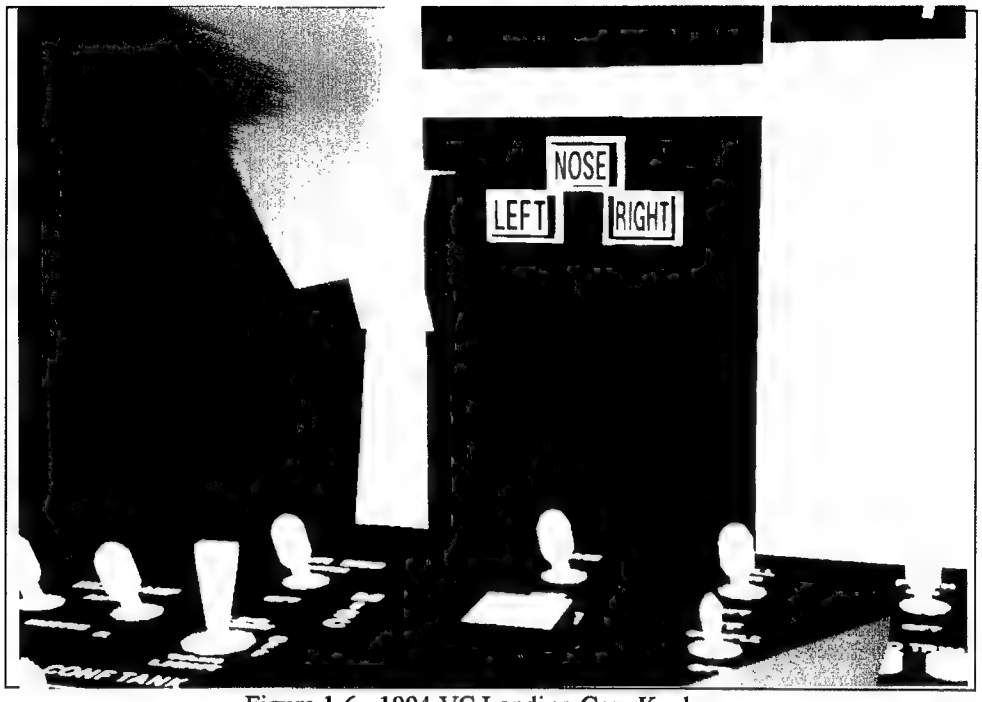


Figure 1-6. 1994 VC Landing Gear Knob.

1.9 Constraint

The major constraint on this research is the aging hardware. The years of VC research have taken a toll on the hardware support tools. The aging Thrust Master WCS Stick, Throttle, and connector cables often need coaxing prior to running the VC. Presently, only one of the five throttles in stock is working. The AFIT Graphics Laboratory's two Silicon Graphics Reality Engines are not useful if the VC support tools are in disrepair. Additionally, one of the Reality Engines has only two processors compared to four processors on the other computer. The two processor machine can not maintain an acccptable frame rate with the features designed into the 1993 VC. Thus, the validation of the 1994 VC must be on the four processor machine.

1.10 Thesis Overview

Chapter II presents an overview of virtual environments and user interface enhancements. Chapter III recounts the requirements and motivation used to assemble the VC. A description of the methodology follows in Chapter IV. The last chapter deals with results of the study and discusses suggestions for future research.

II. Literature Review

2.1 Overview

The Photo-Realistic AFIT Virtual Cockpit (VC) is a virtual reality tool designed as an interactive graphical aircraft simulator. This chapter reviews the literature concerning virtual environments, distributed simulation, VR interface technologies, and the AFIT Information Pod which apply to the VC.

2.2 Virtual Environments

Virtual environments have greatly improved since Ivan Sutherland proposed the concept of a multi-sense user display called the "Ultimate Display" in a 1965 paper to the International Federation of Information Processing Societies (Sutherland). This section presents recent applications that represent the "state of the art" in virtual reality and insight on the direction of future work.

Applications. Virtual worlds have been used in scientific data visualization, information control, simulation, and other applications (Aukstakalnis and Blatner; Bryson; Bryson and Levit; Zelter and Drucker; Erichsen; Cooke and others). The various applications, although having different purposes, have an overlapping common objective which is to simplify the user's interaction with the computer. The two key aspects of VR that researchers are examining are vision and intuitive user interaction. Vision has been emphasized because it is considered the most effective method for presenting large amounts of information with precision. Providing large amounts of information through visual representations is also used for making complex dynamic problems intuitive. By allowing a user to "physically" manipulate the visual presentation, the complex dynamics involved are reduced to relatively simple user commands or movements (Ling).

Scientific Visualization. Steve Bryson of NASA Ames Research Center describes a virtual windtunnel he developed to present the three-dimensional fluid flow dynamics of pre-

computed (on supercomputers) calculations. Within the virtual environment, researchers are able to 'grab' the seedpoints controlling the visualized flow fields. By moving a group of seedpoints around, the researcher is able to identify the interesting areas of the fluid dynamics problems without worrying about the details of the interface (Bryson; Bryson and Levit).

Information Control. A VR planning application differs from a scientific visualization application in purpose and technique. Rather than recreating theoretical, hard to visualize calculations and dynamics, the planning application attempts to simplify great quantities of a dynamic real world situation by presenting the information more concretely through a graphics display. A dynamic planning heuristic also works on the data to provide suggested solutions to the planning personnel. For example, the purposes of a computer-based VR mission planning application are to optimize air assets, maximize the probability of success, determine mission data, weapons loads, schedules, and avionics packages. In addition, the VR planning application provides a tool crew rehearsal. Like the scientific visualization system, the mission planning application is designed to be as transparent as possible, requiring a minimum of computer expertise and programming skill (Zelter and Drucker).

Flight Simulators. Unlike the planning application and scientific visualization tool, the VR flight simulators do not provide the user/pilot with a means of controlling the environment. The purpose of a VR flight simulator is to provide the pilot with a realistic, responsive, and unpredictable simulation. Producing realistic flight dynamics and providing for human opponents via a networked interactive simulation is a recent approach to achieving a high quality and combined force flight simulation. Like the other systems, the pilot's interface with the distributed virtual environment is as transparent as possible (Erichsen; Cooke and others).

2.3 Distributed Simulation

Distributed interactive simulation (DIS) is one of seven critical technologies identified by the pentagon several years ago to receive high-level attention and funding. The goal of DIS is to create a sophisticated VR with capabilities far exceeding those of any single facility. To

accomplish this goal, the Defense Simulation Internet (DSI) is being developed as a world-wide integrated network for global simulation (DIS, Stytz and others; Institute).

According to Bess, the key networking concept stipulates that all players on the network perceive the same events. All network players must have intervisibility, see the same atmospheric and illumination conditions, and use realistic sensor simulations in order to achieve a meaningful simulation. For example, if only one player can see flares, then the value of the networked tactical training is lost (Bess). Similarly, if players behave unrealistically, the simulation's value is also lost. The tactical training nature of simulation influences the application as follows: (Bess)

1. Preserve intervisibility
2. Own vehicle freedom of movement
3. Natural vehicle dynamics
4. Able to take part in extended scenarios
5. Full use of crew positions
6. Full sensors, sight, and view points
7. Full use of all weapons
8. Complex area content or density
9. Dynamic database changes
10. Unpredictable dynamic scene content

The several distributed virtual environment applications being developed at AFIT are influenced to a limited extent as described by Bess. The Satellite Modeler (SM) and Synthetic BattleBridge (SBB) rely on the AFIT Information Pod (discussed below) to move through and interact with the VR environment (Vanderburgh; Rohrer; Kestermann). Because the SM and SBB user's move within the VR as non-player observers, the first seven influences above do not affect the SM and SBB applications. However, because the satellites controlled by the SM's user are network players, the Bess' influences do apply to the satellites. As a network player, the VC is also affected by the factors listed above.

2.4 *Virtual Reality Interfaces*

The modern-day technological explosion has extended the potential of computers to expand human information management. However, the interface employed largely affects the degree to which computers can be used as tools (Furness and Dean). Work centered on putting the user within the VR has emphasized technologies such as the head mounted display (HMD), data gloves, and position trackers. Butterworth, and others have integrated an HMD and data glove in an effort to provide a more intuitive computer aided design (CAD) modeling interface. According to Butterworth, "even a novice user can understand how to manipulate a model by reaching out and grasping" (Butterworth and others). For three dimensional interaction, an evaluation of several input devices concluded that multi-dimensional devices like the data glove and the Spaceball were easier and faster to use than the two dimensional devices (Venolia). VR systems focus on stimuli for the visual sense and provide some method to navigate within the system. More progress towards the Ultimate System is possible with tactile feedback devices in the prototype stage (Astheimer and others). Research advances in multi-dimensional interaction are still in their infancy, just having begun the same development process as was applied to the two-dimensional interface ten years ago (Figueiredo and others).

2.5 *AFIT Information Pod*

The AFIT Information Pod is a paradigm originally developed as an interface for AFIT's SBB and SM applications. The Pod is composed of two interfaces: 1) a window-on-the-world view that assumes the presence of a CRT and a mouse and 2) an immersive interface designed to support user interaction when the user's view is restricted to the virtual environment. The immersive interface assumes the user will use an HMD or a large screen surround display as well as input devices such as the a DataGlove™, 3D mouse, 2D mouse, spaceball, or BioMuse™ controller (Stytz and others). The AFIT Information Pod provides an application independent tool for communicating with and/or gathering information from the players in the VR environment.

The Pod consists of one or more three dimensional control panels located around the user's view point in the virtual environment. The number of panels depend on the application. For example, the SBB, a distributed virtual environment platform that allows users to monitor and assess the activities within the virtual battle space, uses five control panels. The SM uses three panels to accurately portray satellite and planetary motion. Tailored to the application, each control panel is separated by function (Kestermann). The three panels within the SM allow the user to move the view to any point in the environment; to attach to a moon, planet, or a satellite; and to toggle the visual display options (Vanderburgh).

Each of the AFIT Information Pod's panels may contain one or more sub-panels. Sub-panels are inactive and invisible panels which are "stacked underneath" the primary panels (Kestermann). These sub-panels become active when the conditions set in the application are met. For example, the SM sub-panel for controlling satellites become active when the user attaches the SM's Pod to a satellite (Vanderburgh). The user inputs commands into a panel through the buttons on the panel's surface.

Buttons allow the user to interact with the environment, to request information, and send commands into the VR. Although other methods of movement are examined, the varied degrees of motion required by the two SBB and SM are achieved by the button paradigm. Of the other options examined, Kestermann states: (Kestermann)

> "Unlike the haptic or voice control paradigms, there is no need to memorize a list of commands such as specific hand gestures or voice commands. The spaceball and joystick, although considered by many to be intuitive and easy to use, do not support all the degrees of movement required by the pod."

For activating panel buttons, Kestermann uses a mouse. The mouse provides a familiar and well developed interface. The data glove is intuitive in concept, but jitter due to interference, tracking inefficiencies, and implementation difficulties make the glove impractical as one of the Pod's interface devices (Kestermann).

2.6 Conclusion

The VR applications described in this chapter indicate the many possible directions future VR simulations may take. For example, a tool like the AFIT Information Pod can create a rippling effect across a wide field of VR research. DIS technology presents the potential to expand VR applications by allowing large scale realistic simulation. To interact within these VR environments, researchers have concentrated on multi-dimensional input devices to compliment display tools such as the HMD. Until the interface technology becomes fully mature, however, methods for filling the gaps in the multi-dimensional technology will be used just as was done for two-dimensional technology during its development (Figueiredo and others). The AFIT Information Pod uses a 2D mouse input device because the 3D devices available are inadequate for the applications.

III. Requirements and Motivation

3.1 Overview

This chapter details the goals, requirements, and motivation of the VC's photo-realistic virtual environment. In particular, the VC's models, Head Mounted Display (HMD) and CRT interfaces, and mouse and keyboard interfaces are discussed.

3.2 Detailed Photo-Realistic Models

The modeling goal for the photo-realistic VC is the complete recreation of the forward crew station of the F-15E in the virtual environment. The sub-goals for the model are modularity and accuracy.

A modularly structured cockpit model for the photo-realistic forward crew station reflects a hierarchy similar to the systematic arrangement of the cockpit control surfaces found in an F-15E (Aeronautical). The F-15E's forward crew station has three large structures: 1) main instrumentation panel, 2) left console, and 3) right console. Each structure listed contains functionally separated sub-structures such as the warning lights, the multipurpose displays, the dials, and the control panels. By generating models at the sub-structure level, future changes, such as the environmental control panel modification, to the F-15E can be incorporated into the VC by replacing complete models.

The VC's panels will take advantage of redundant parts to simplify the modeling process. Many of the switches and knobs in the F-15E are identical and need only be modeled once. For example, the sensor panel on the forward station's left console has three toggle switches and four knobs. In all, the left console's panels have more than thirty switches and knobs, including those on the sensor panel. Thus, one copy of a toggle switch can be used four times on the sensor panel, five more on the rest of the left console, and many more times on the right console and forward instrument panel.

MultiGen will be used to generate the models in a modular hierarchy. At the top level of the hierarchy is the complete forward crew station - the left console, the right console, and the main instrument panel models (Figure 3-1a & b). Each of the crew station's components is divided into its sub-components such as the sensor and air regulator panels. These sub-components are the fundamental levels at which a forward crew station will be constructed. Incorporating modularity into the models will allow real world design changes in the F-15E to be dropped into the VC.

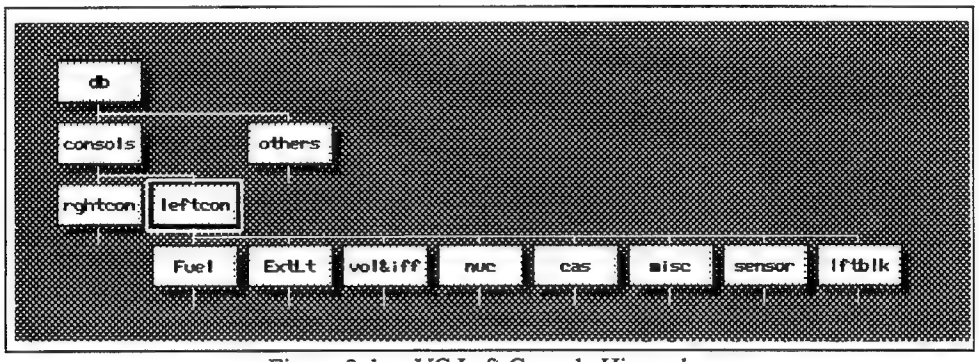


Figure 3-1a. VC Left Console Hierarchy.
The "consols" group is the parent node to the lftcons group. The left console's panels are divided according to function.

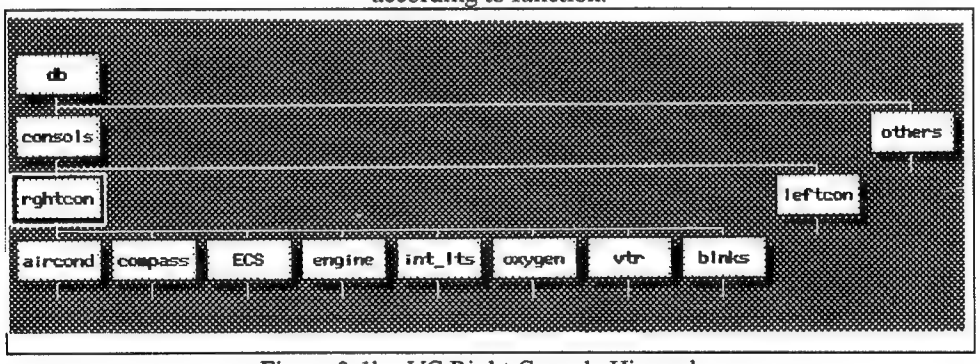
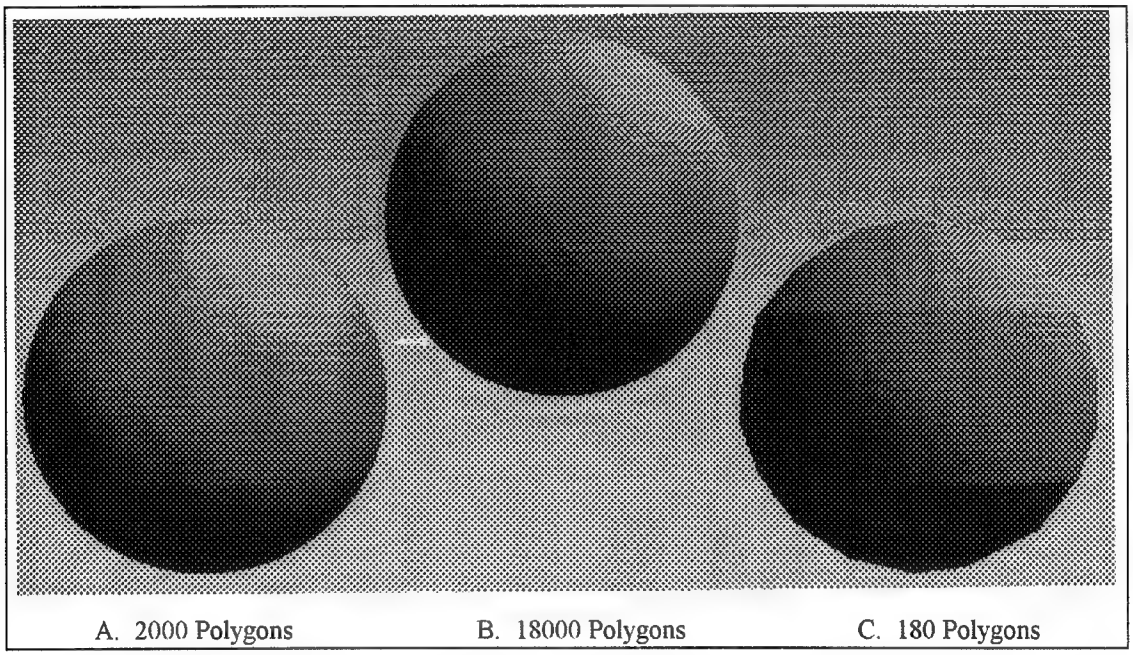


Figure 3-1b. VC Right Console Hierarchy.
Similar to the left side's structure, the panels models are divided according to function. Note: the "others" node on the right refers to the instrument panel's parts, such as the warning lights.

Like modularity, generating accurate models is also a sensible goal. The level of accuracy necessary to adequately model the interior is dependent on the simulation's purpose and the processing power of the computer running the simulation. For the VC, the photo-realistic virtual environment requires the most detailed models the hardware can support. If the VC had no other requirements, a highly realistic model could use tens-of-thousands of polygons to recreate the

forward crew station. However, the model detail is in direct competition with other traditional drivers of flight simulator and VR technology. Traditional drivers the VC must also satisfy include: 1) mathematics involved with simulating the physics of flight, 2) collision resolution techniques used to interact with other entities in the virtual environment, 3) complexity of the environment (outside the cockpit), 4) human factors and interface design issues, 5) VR players, and 6) network interaction requirements. Figure 3-2 depicts 3 balls of varied levels of complexity. Table 3.1 depicts the frame rate impact of adding each ball in the VC simulation.



A. 2000 Polygons B. 18000 Polygons C. 180 Polygons

Figure 3-2. Three Balls.

Ball A is rounded, yet the shadows highlight some edges on the surface. Ball B has a round shape and a soft realistic shadow. Ball C has a pointy silhouette and distinct highlights on its surface.

Table 3.1 Ball Complexity and Frame Rate Impact

	Ball A	Ball B	Ball C	No Ball
Polygon Count	2000	18000	180	0
Frame Rate (Hz)	12.5	9.5	15.6	15.0
Net Impact	-16.7%	-36.7%	+3.3%	0%

The numbers shown in Table 3.1 are highly dependent on the scene being displayed. For example, Balls A and B have less impact on a more complex scene. In the case above, inserting Ball C into the VC improves the performance because the ball is less detailed than the instrumentation it

blocks. Table 3.1 also illuminates the conflict between a model's complexity/realism and its processing impact. Consequently, the VC's models must look realistic and, yet, be simply constructed to avoid disrupting the simulation's frame rate - a pair of contradictory goals. Fortunately, the pilot's interface into the virtual environment can make up for some of the modeling short-cuts used as will be discussed in Section 3.3.

3.3 Head Mounted Display and CRT Interfaces

The photo-realistic VC will include both a head mounted display and a CRT view - a window's eye view - of the virtual environment. Both interface types have been demonstrated in previous versions of the VC. However, in this implementation, the purpose of each interface will be separated.

The photo-realistic VC's interface sub-goal, to provide an HMD interface as the pilot's primary view into the virtual environment, has two objectives: 1) improve the simulation's flight characteristics and 2) simplify interaction with model controls. Improving flight characteristics refers to the pilot's quality of flight and situational awareness. For instance, by using an HMD the pilot is able to 'look ahead' as he/she turns the aircraft or to keep another aircraft in sight longer during a fly by. This first objective has been demonstrated in the previous VC 2.0 (Erichsen). The second objective, to simplify the interaction with the photo-realistic control systems in the VC, will be demonstrated in the photo-realistic cockpit by the enhanced perception of three-dimensions due to parallax (Zeltzer and Drucker).

As a complement and a contrast to the HMD interface, a window's eye view will also be used in the VC. The CRT has long been the standard viewing device for VC similar to the familiar view in countless video game flight simulators. In addition, the CRT allows useful features such as simplicity of display and a non-restrictive viewing device. Using the window view and the HMD view into the VC will demonstrate the functionally complementary features of each interface.

The HMD and window's eye view interfaces differ from each other with respect to the type of freedom each allows. A pilot in an immersive, fully functional, photo-realistic environment

benefits from the freedom of view an HMD provides within the virtual world. While the window view restricts the free movement of the user's view, the window view into the VC provides freedom of movement in the real world. Two possible uses for a window view are as an over-the-shoulder view for an instructor or for an exercise monitor.

3.4 Mouse and Keyboard Interfaces

The photo-realistic VC also includes methods for users to interact with the knobs and switches modeled in the virtual environment. The 1994 VC uses both a mouse and a keyboard interface.

The mouse interface is the method a pilot uses to select the switches and knobs in the VC's forward crew station. The mouse offers practical and human factors advantages over the data glove and the keyboard interfaces available in the AFIT Graphics Laboratory. From a practical perspective, the mouse is a common interface available on home computers and work stations; its movements and uses are well understood. Also, the mouse hardware is reliable, seldom requiring maintenance other than cleaning. From a human factors perspective, the mouse requires less concentration than the data glove to maneuver over the flat surfaces depicted in photo-realistic VC. Although, Butterworth and others assert that the glove is superior to the mouse, the latter has two advantages over the DataGlove for this specific application. First, the mouse is a more mature input device which has enjoyed widespread use since the early 1980s (Figueiredo and others). Second, by limiting the button selection to two dimensions within the virtual environment (i.e. the surface of a console) the problem of selecting a button becomes tailor made for the mouse. As the glove matures, advances such as tactile feedback (Astheimer and others) should facilitate a three-dimensional approach to the button selection problem. The mouse is easier to use with the HMD versus the keyboard. Commands are easily executed by pointing and clicking because the mouse cursor remains in the user's field of vision. On the other hand, if the user is wearing an HMD, the keyboard is blocked from view. Consequently, the user must search for the correct keys to issue a command.

In the VC, the purpose of the mouse interface differs from that of the keyboard interface. While the mouse provides a means for the user to interact with the virtual cockpit, the keyboard offers controls which affect simulation features. The separation between functions also highlights the operational use envisioned for the VC, for example, a pilot wearing an HMD will fly the VC and a controller or instructor can oversee the simulation from a workstation.

3.5 Conclusion

The photo-realistic VC research centers on increased realism and control for a pilot user. The models re-create the forward crew station, the HMD provides an eye into the cockpit, and the mouse provides a hand for the pilot to interact with the environment he/she sees. The methods used to generate the environment will be detailed in the next chapter.

IV. Design and Methodology

4.1 Overview

The overall design objective for the 1994 VC is to provide a more realistic simulator environment on a relatively low cost machine while retaining the previous version's large force, DIS network compatibility. To achieve the VC's major objective, elaborate models of the interior of the cockpit need to be constructed. Once the models are finalized, a method for communicating the pilot's commands will be integrated along with the capability to translate the pilot's commands into corresponding actions. This chapter details the model construction and simulation interface design of the improved VC.

4.2 Model Baseline

The first step prior to the VC's cockpit model construction is to establish a baseline configuration. The baseline, to provide the data needed for a realistic final product, should meet the following criteria:

1. Based on F-15E instrumentation
2. Provides dimensions of equipment
3. Provides information on cockpit ergonomics
4. Provides functional information of controls
5. Provides available color and text information

Three sources, the F-15E Human Factors Documentation (Aeronautical), two photographs (Figure 1-4; 4-1), and panel measurement made at the Human Factors Laboratory (Kriss and Kubiak), will be used as the baseline. The F-15E Human Factors Documentation contains the 1988 F-15E configuration, ergonomics, and functional data (Aeronautical); the photographs provide color information; and the panel measurements will allow the models to be drawn to scale.

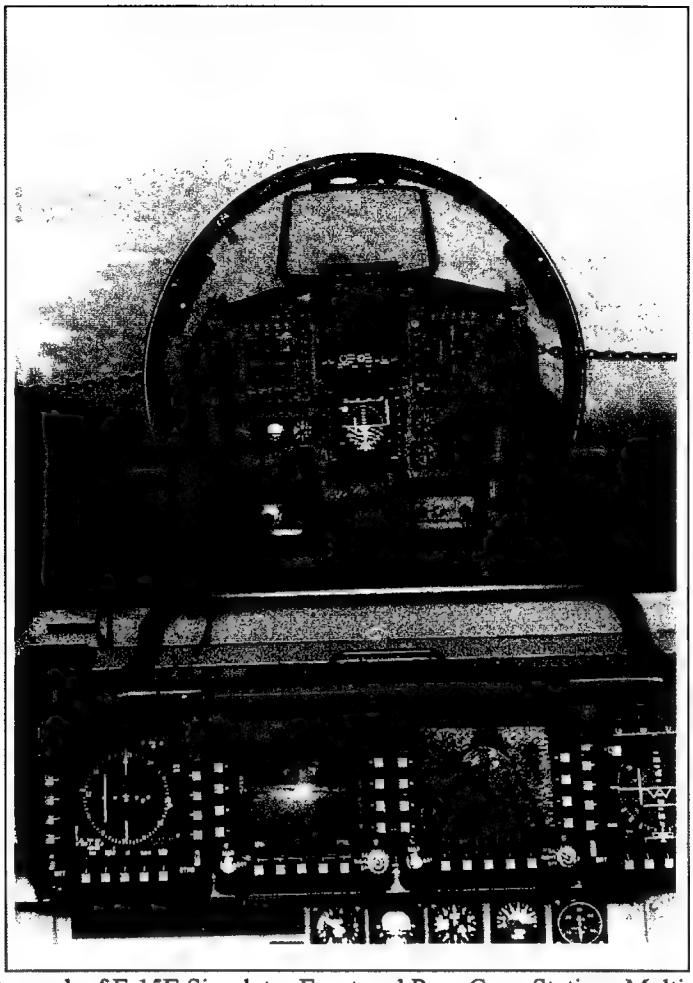


Figure 4-1. Photograph of F-15E Simulator Front and Rear Crew Stations Multipurpose Displays.

After the baseline is selected, the next steps are the construction of polygonal representations, applying texture maps, and placing the models within the VC's F-15 MultiGen model. The following section describes the model construction and how the advanced modeling tool, MultiGen, was used.

4.3 Model Construction

Scale. Having the correct three dimensional measurements for all the equipment is needed to make models that "look" right. While the schematics (Figures 4-2, 4-3, 4-4) and the photographs provide a clue about relative size of panels, none of them include any measurements. However, a scaled model can be generated with the relative information in the baseline

documentation and with measurements for one or two panels. Fortunately, the Human Factors Laboratory's F-16 cockpit mock-up has two pieces of hardware, the clock dial and the air regulator panel, that are identical to the F-15E based on the schematics available. According to Kriss and Kubiak, the Human Factors Laboratory uses a similar approach to constructing cockpit mock-ups when the dimensions are not available (Kriss and Kubiak). In addition to the panel measurements, the Human Factors Laboratory also has a stock room with the switches used on the Air Force jets. With this information, scaled models of the forward components can be built and inserted into the F-15E model utilized by the VC (Figure 4-5).

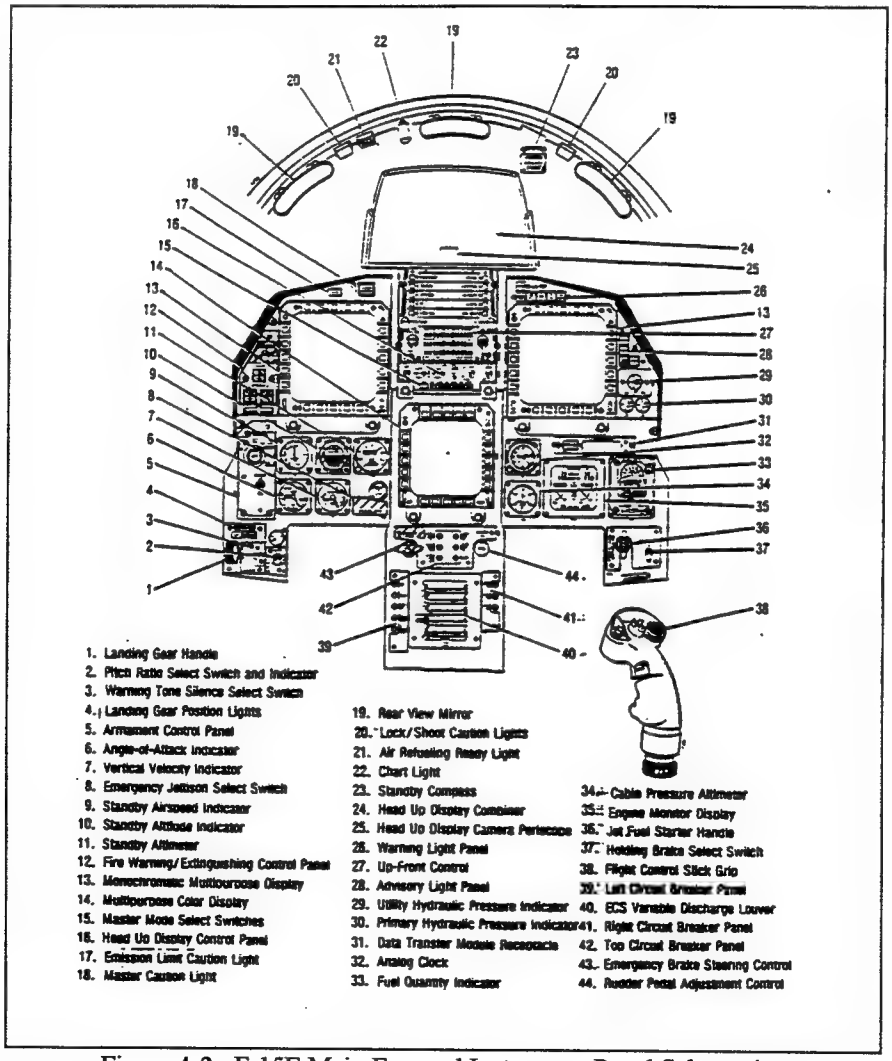


Figure 4-2. F-15E Main Forward Instrument Panel Schematic.

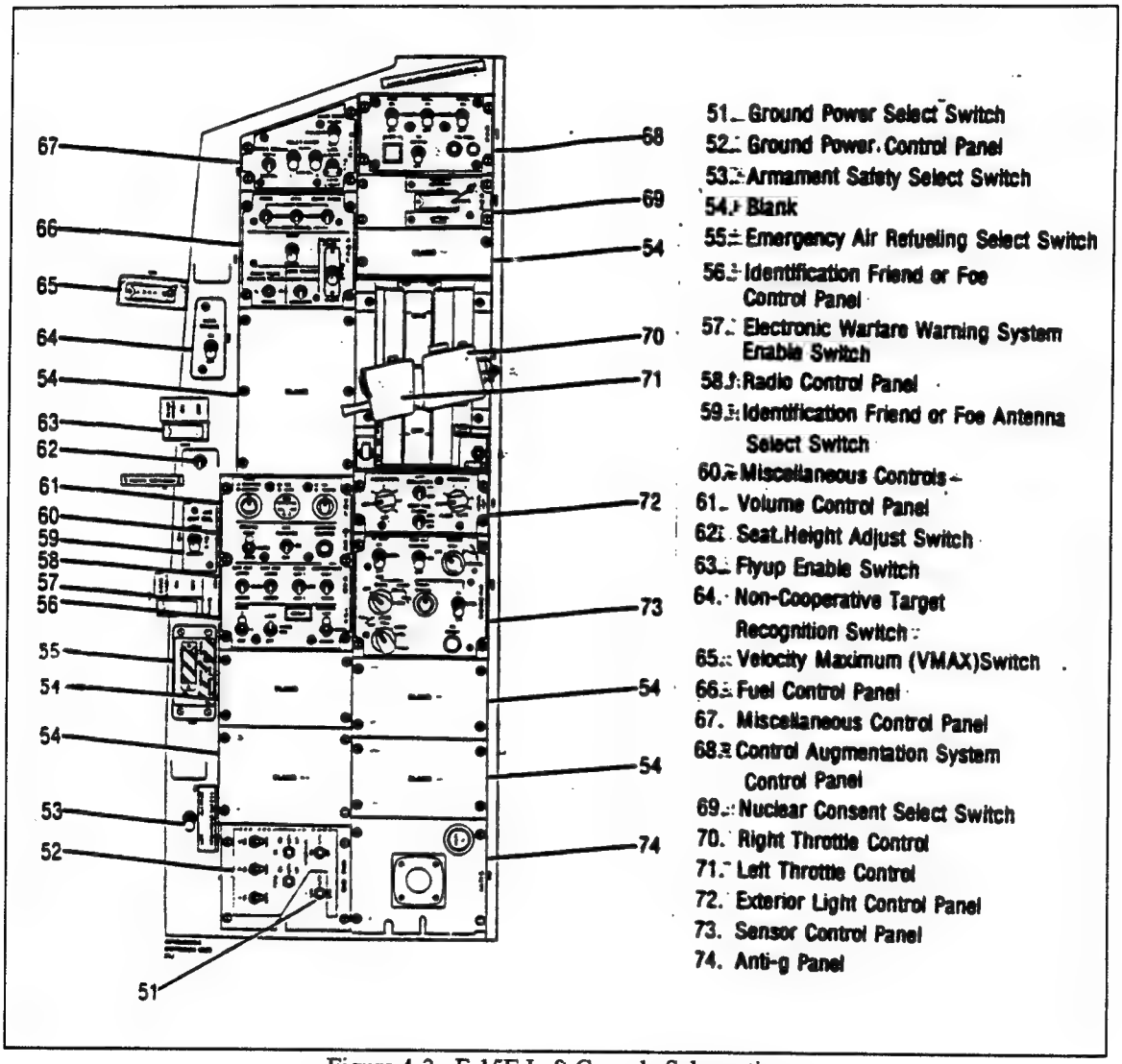


Figure 4-3. F-15E Left Console Schematic.

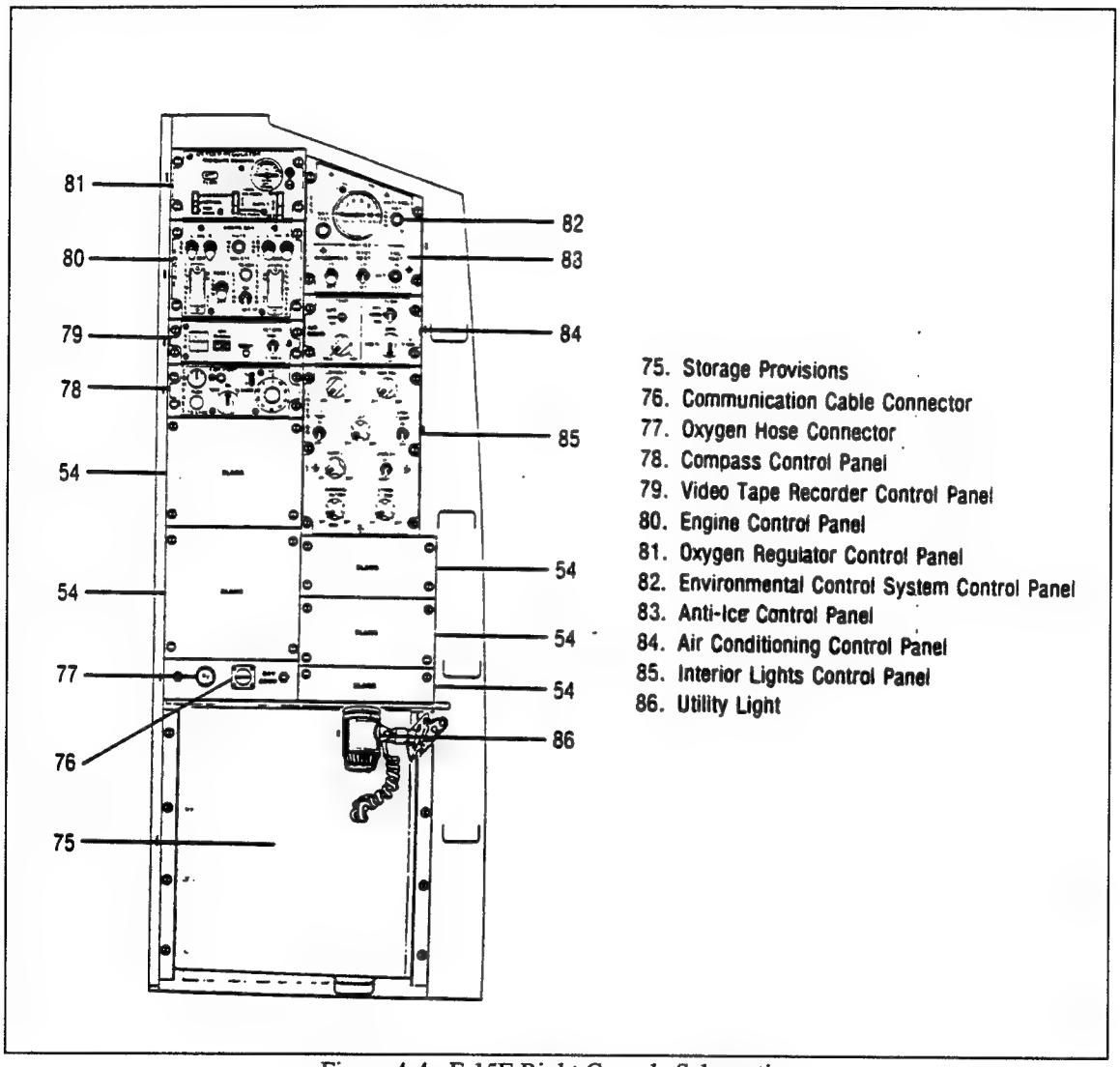


Figure 4-4. F-15E Right Console Schematic.

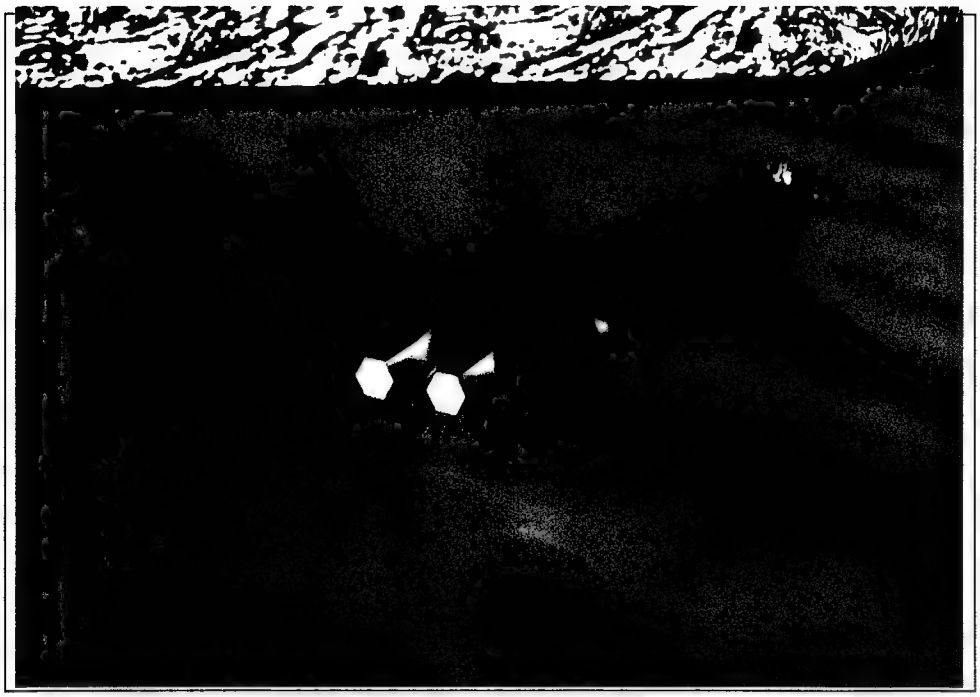


Figure 4-5. F-15E model used in the VC is a MultiGen Flight model.

Modeling. This research employs three approaches to generate the photo-realistic VC models. The first approach use a texture map of each complete panel to create the photo-realistic models. The first approach and why it fails will be dealt with in the next section. A discussion follows on the second approach, building models within the F-15E model file, and the third approach, building models independent of other model files.

The second approach, building models within the F-15E model file and using limited texture maps, appears attractive because the photo-realistic models would be drawn to scale and in place with respect to the F-15E model's origin. However, this approach proves to be unmanageable because 1) F-15E model is too complex and 2) MultiGen grid spacing restricts the minimum size of the photo-realistic models.

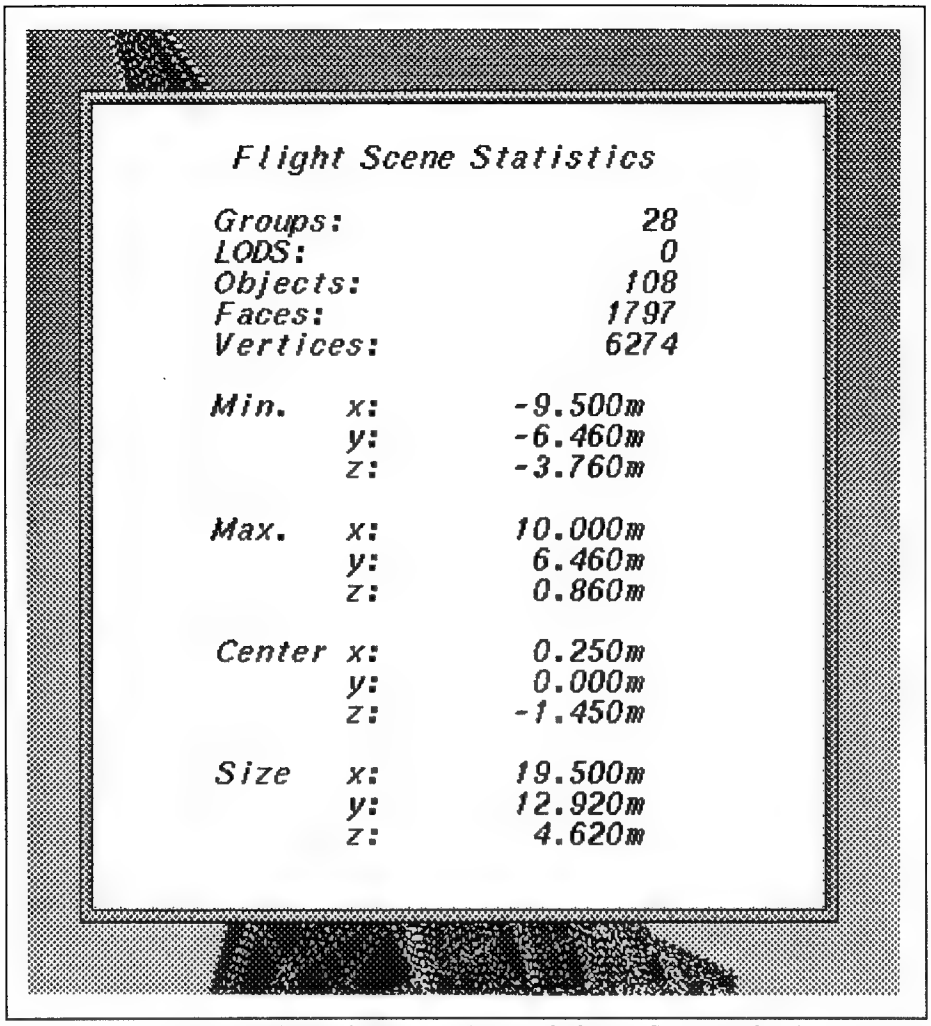


Figure 4-6. MultiGen Flight Scene Statistics Display (Software Systems) for the F-15E Model. The statistics table provides a measure of the models complexity. This model is made up of 28 Groups and 108 Objects. More important statistics when considering the rendering speed of a simulation is the number of Faces (polygons), 1797, and the number of Vertices, 6274. Note: Although this model are complex, the 3.49:1 ratio of vertices to polygons means that the modeler attempted to keep faces as simple as possible. This will simplify the model in preparation for the rendering application.

The F-15E model's complexity, (Figure 4-6) is a problem because the Silicon Graphics 4D which has MultiGen installed does not respond within a reasonable amount of time to simple modeling commands. Changing the view point, for example, takes at least half a minute. The incremental changes in view that MultiGen allows becomes unbearable because after each increment, the computer would freeze until the model is refreshed. To understand the second problem, the MultiGen grid must be explained.

The grid is a MultiGen feature that normally appears on either the XY, XZ, or YZ planes and can be adjusted to any other orientation a modeler desires. Also, the grid spacing can be initialized to inches, feet, meters, or kilometers depending on the size of the model being generated. Once the spacing is set, the smallest units the grid displays is one hundredth the spacing setting. For the VC's F-15E model, the grid spacing is set to a meter. This limit makes the generation of simple models like the toggle switch nearly impossible because the grid's minimum resolution was greater than the width of the toggle switch. Fortunately, the unmanageable database and incompatible units of measure problems reveal themselves early. Because this approach does not prove promising, it is abandoned in favor of the third method, separately generating models.

In the third approach, by modeling the consoles separately, the problems encountered in the second approach are avoided. Using the forward crew station diagrams, the left and right consoles' footprints are mapped into a MultiGen space. The MultiGen tool's grid spacing is set to one meter and a 1 meter = 1 inch conversion is made when modeling the consoles. The 1 meter = 1 inch scale allows the model's resolution to be one hundredth of an inch. For example, the Oxygen Regulator Panel width, 5.25", is drawn as 5.25 meters in the modeling space. After each footprint is established, the panels are cut out in cookie cutter fashion using MultiGen's polygon slicing command. The cut panels are then copied into separate files. Once divided, model switches and knobs are added to each panel's data base.

The instrumentation of the F-15E forward crew station is filled with dozens of different shaped toggle switches and knobs. For example, the Engine Control Panel has four types of toggle switches and the Sensor Control Panel (Figure 4-7) has four different shaped knobs. Among the reasons cited for the difference in shapes is the tactile feedback each switch provides (Kriss and Kubiak). The various shapes aid pilots by allowing them to feel for the switches and knobs they need. Thus, the pilots can accomplish minor tasks while maintaining their look up time or situational awareness.

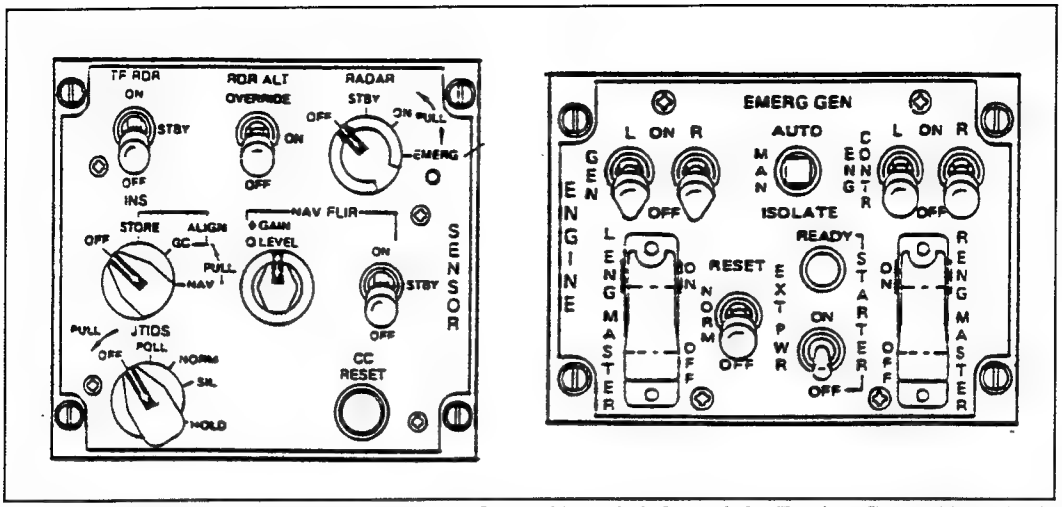


Figure 4-7. Schematic Diagrams of the Sensor Control Panel (left) and the Engine Control Panel (right).

In the VC's completely virtual environment, despite not having tactile feedback, modeling the correct shapes remains important for providing visual cues. Many of the knobs in the cockpit, while similarly shaped, differ in the texture of their surface; for example, some knobs are smooth while others are ridged. Because modeling all of the knobs perfectly would greatly complicate the models and the data base, only two types of knobs will be modeled in this version of the VC. Although, this short cut will slightly detract from the final product, the size and shapes modeled will be a reasonable representation of the actual knobs present.

Texture. The first approach to generating photo-realistic displays uses textures as a short cut to avoid complex polygonal models. The short cut, however, is impractical for reasons described below.

The short cut to generating a texture map attempts to create a texture file using a photograph or schematic from the baseline documentation. At first, this approach is attractive for the following reasons: 1) only a single polygon is needed, 2) the panels can be mapped to scale based on the texture map, and 3) the correct positions of moving parts are also mapped onto the polygon. This approach is later found to be impractical due to memory and resolution problems. The first attempt to generate a memory map is from a photograph (Figure 1-4) using Macintosh Adobe PhotoShop and a scanner. Potentially, a realistic texture can be taken from the picture. However, the resolution and contrast of the pictures in the baseline are too poor to be well scanned.

Also, the sizes of the image files resulting from the generated texture files are well over a megabyte each. An attempt is made to scan the panel schematics piecemeal and paste them on their polygonal counterparts. This variation of the texture method does not work because the individual panel bit maps uses too much memory, about 100 Kbytes per mapped panel. Additionally, most of the areas being mapped are the unneeded black areas between the words.

In order to reduce the bit map files and to improve the quality of the text on the panels, new texture map files are generated with text only. The texture map files are converted from Macintosh Adobe PhotoShop GIF files to intensity (int) and rgb files which are compatible with MultiGen; in most cases, a gray scaled intensity format is sufficient. The texture maps contain all the texts on the consoles and main instrument panel (Figure 4-8). Words from the texture files are individually applied to polygons and finally placed in their appropriate positions on the panels (Figure 4-9).

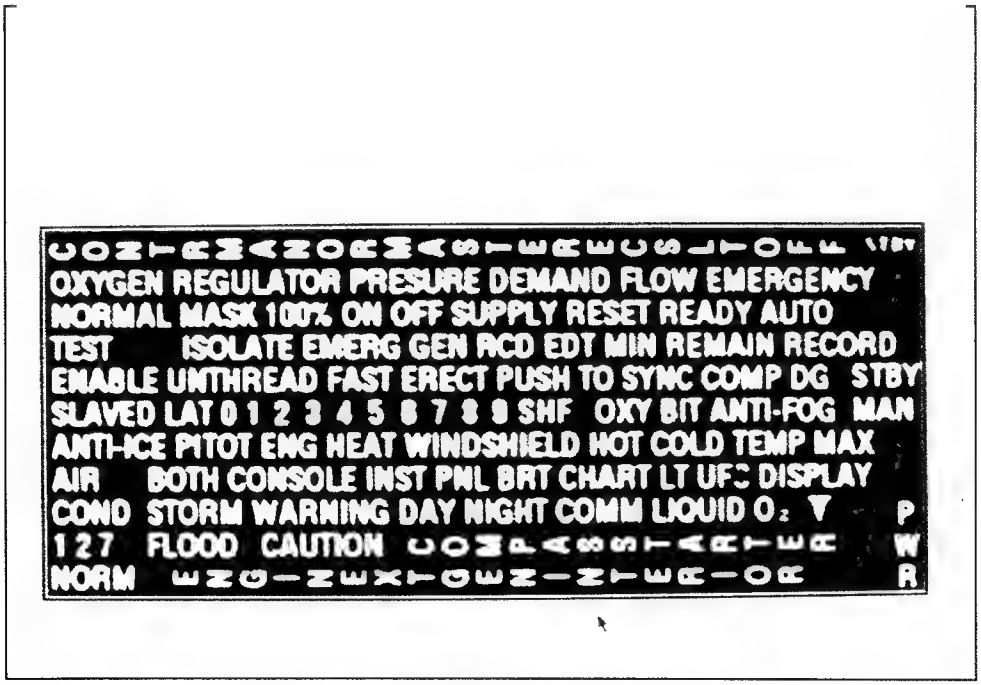


Figure 4-8. Texture File Used for Right Console.
Note: the text is all one size, the size differences seen on the console are due to the size of the polygon the text is mapped onto. A large polygon gives the appearance of greater font as can be seen in Figure 4-9.

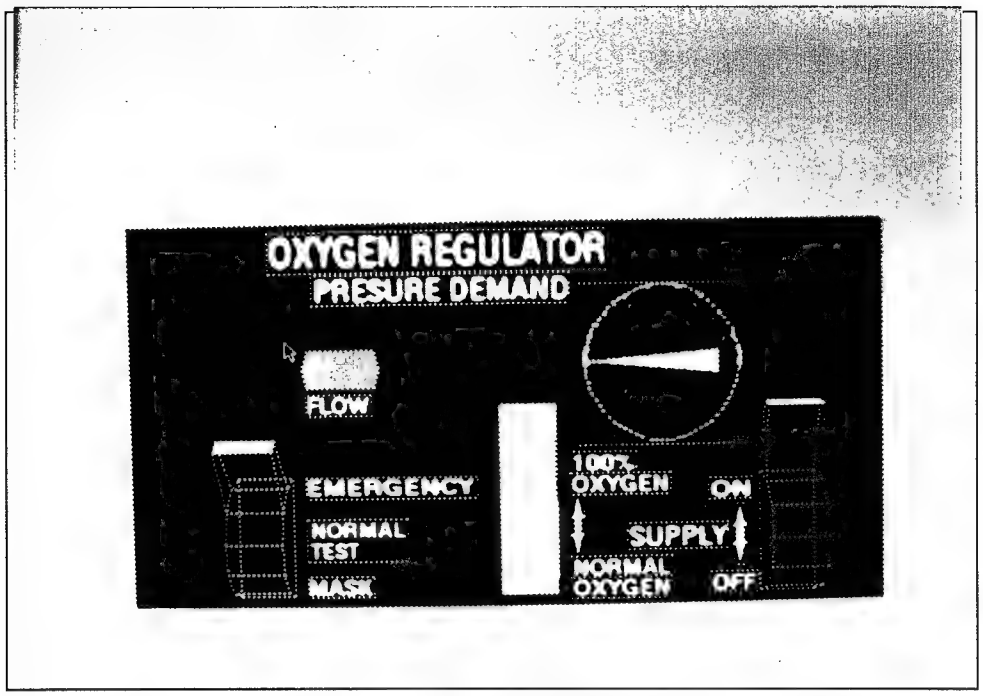


Figure 4-9. Oxygen Regulator Panel.
The polygonal construction is highlighted by the white dotted lines.
Note: Text is scaled to the polygon on which it is applied.

Completing the Model. To complete each console and the main instrumentation panel, each of the component panels and instruments are completed individually. Once the individual pieces are complete, the whole console is assembled (Figure 4-10). After both consoles are accomplished, the crew station is scaled to the same proportions as the F-15E model used by the VC and moved to the correct location within the model. The product at this point in the research is a photo-realistic, non-interactive forward crew station. The next section details the methodology for providing a user interaction with the crew station.

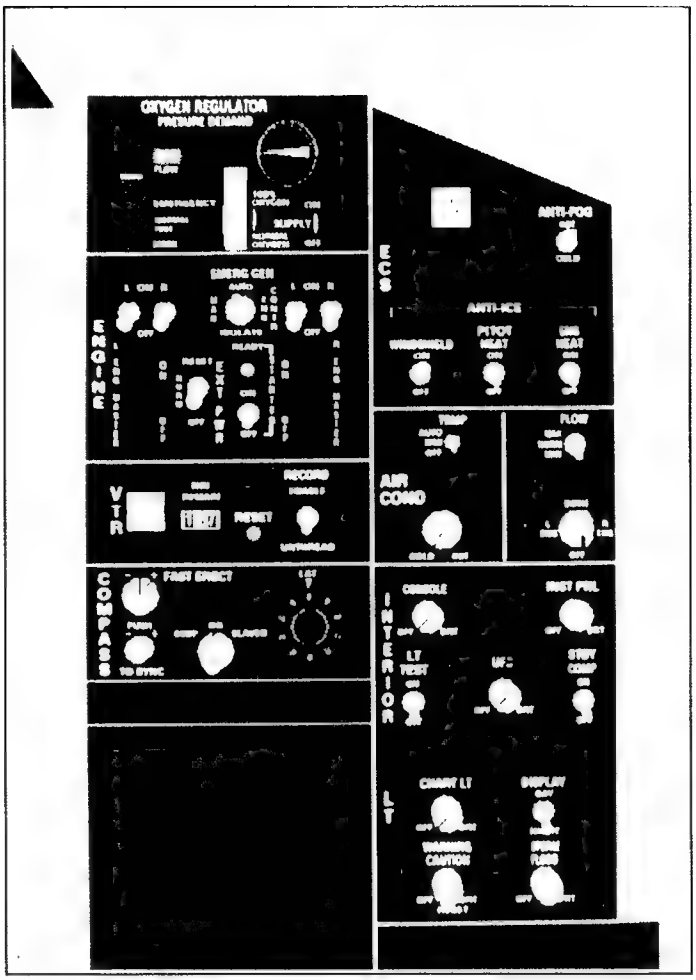


Figure 4-10. Completed Forward Crew Station Right Console

4.4 Switch Movement

The visible movement of switches in the simulation gives the pilots feedback to their input. The position of the static models is defined by a static coordinate system (SCS) matrix. Once defined, static models can not have their position or parts modified (Figure 4-11). After the static photo-realistic models are placed in the VC simulation, the static switches are removed from the models (Figure 4-12). Then, movable switch models are placed, using a dynamic coordinate system (DCS) matrix to define their location in the simulation, where the static models were removed (Figure 4-13). The DCS defined models can be moved by modifying the DCS matrix.

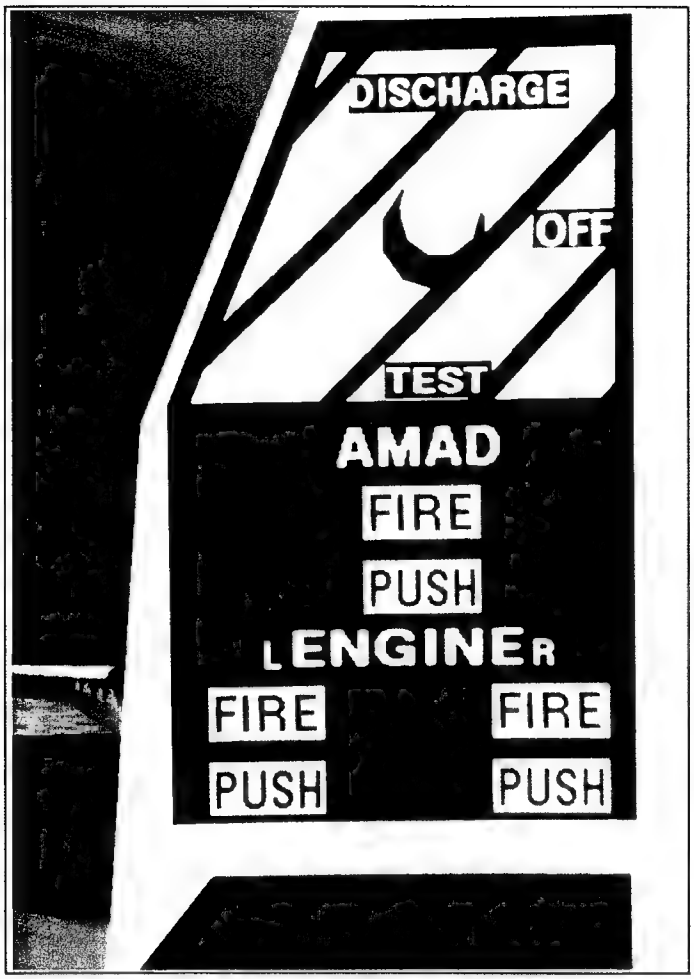


Figure 4-11. Static Model of Fire Warning / Extinguishing Control Panel.

Switch movement in the virtual environment is simulated by modifying a model's DCS matrix then redrawing the model. A smooth motion may be displayed by projecting an incremental series of discrete position changes. However, when modeling the movement of a switch, knob, or light, only the new position (or status) holds any value for the pilot. Thus, the final position is modeled rather than modeling the motion with a series of position changes.

The position changes of each model are based on the type of model being toggled. Each of the switches, knobs, and warning lights has a different type of movement. For instance, toggle switches are rotated about the x-axis, perpendicular to the center line of the switch. Knobs rotate about the z-axis, the center line of the knob. Warning lights turn off and on by translating the textured polygon out of and into the pilot's field of view (Figure 4-14 and Figure 4-15).

Switches and knobs may have more than the two positions on and off. In the photo-realistic VC, the range of motion for the various knobs and switches is simulated by toggling through a case statement. The software keeps track of the knob's current position and of it's next position.

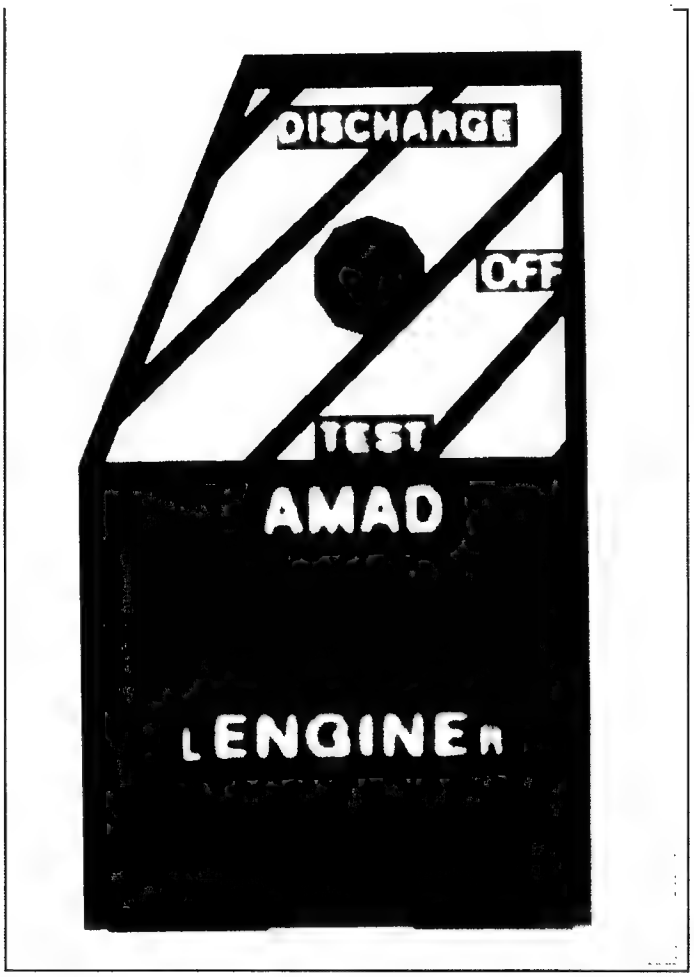


Figure 4-12. Fire Warning / Extinguishing Control Panel without dynamic switches and lights.

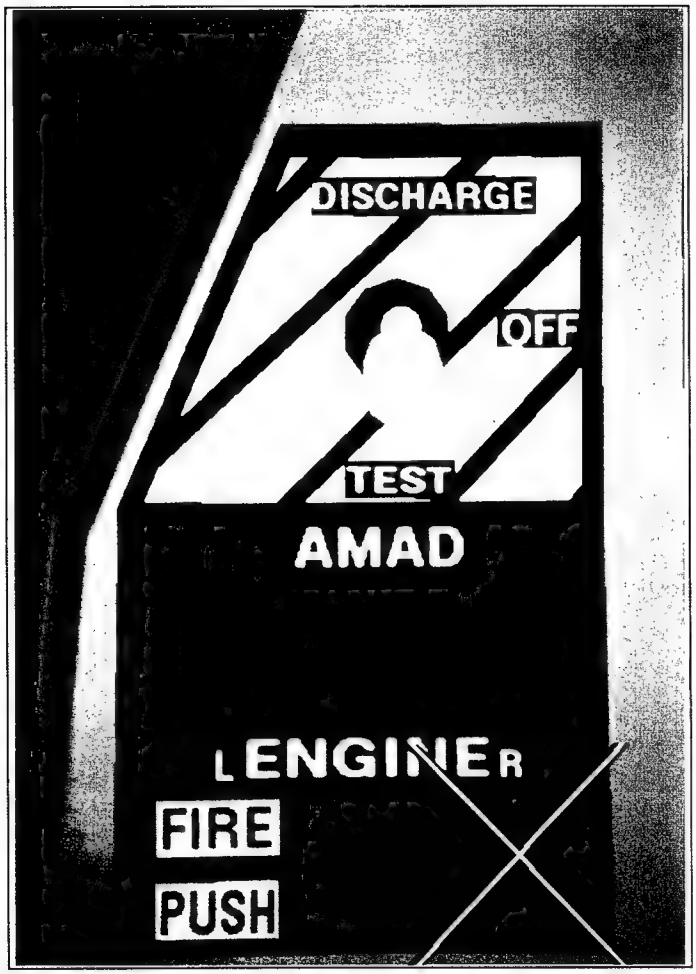


Figure 4-13. Fire Warning / Extinguishing Control Panel with dynamic switches, lights, and cursor. The red outline along the left hand side of the Main Instrument Panel indicates that this is the mouse-active panel.

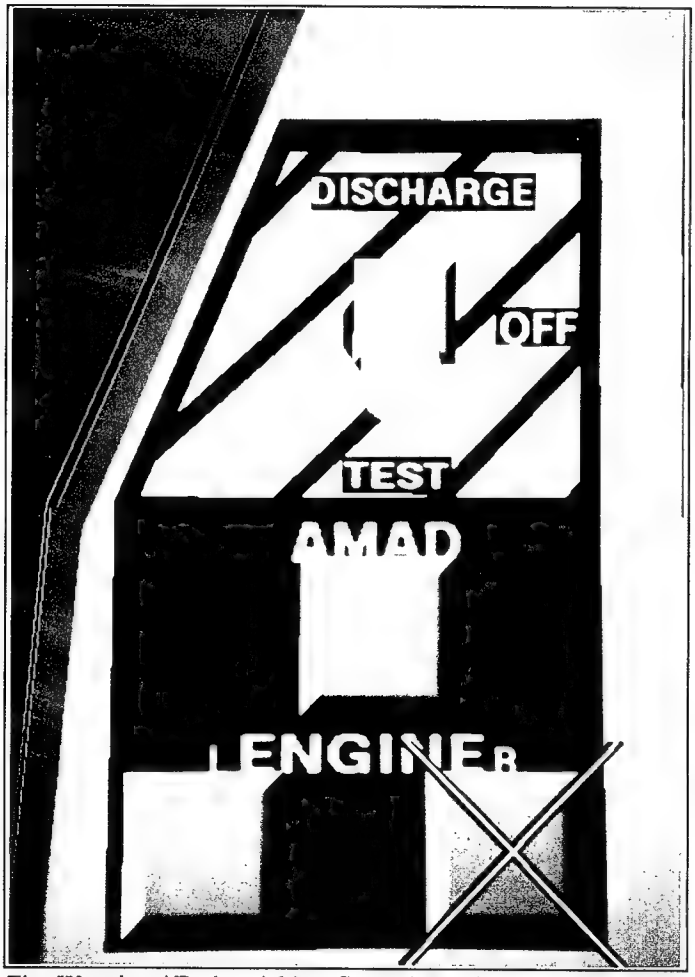


Figure 4-14. Fire Warning / Extinguishing Control Panel with toggling buttons displayed.

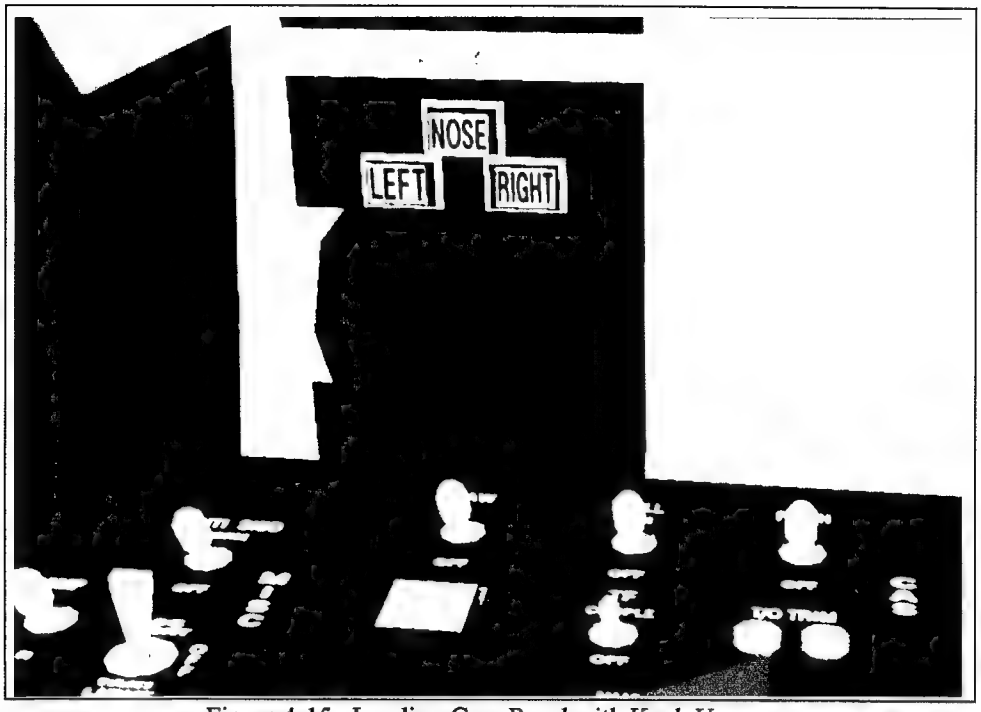


Figure 4-15. Landing Gear Panel with Knob Up .
The gear knob toggles from the up to down position. The gear lights are two sided polygons with green background on the opposite side.

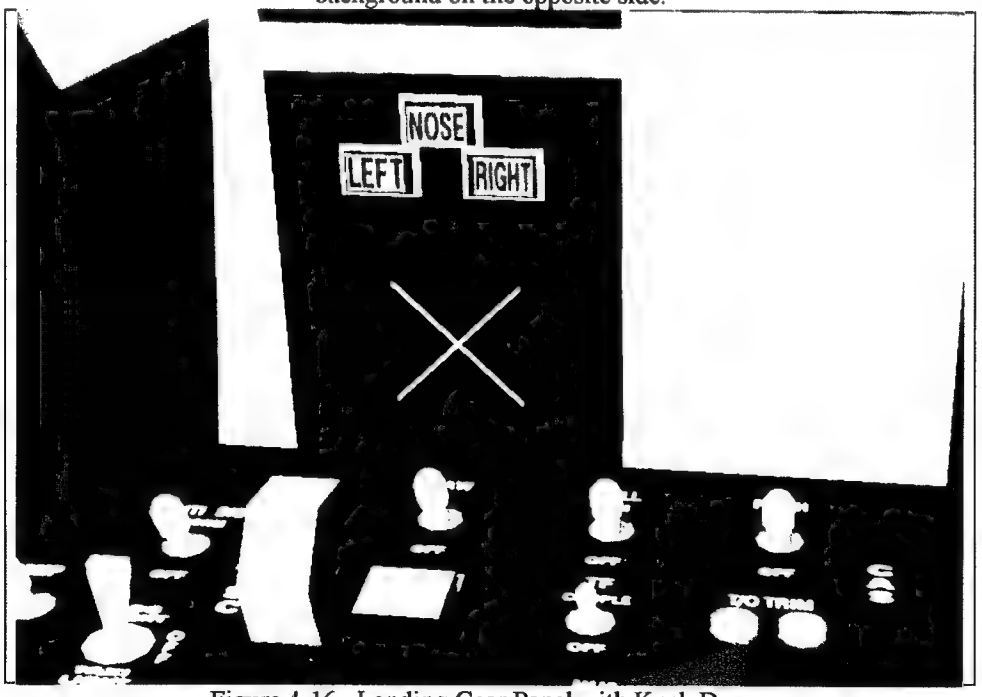


Figure 4-16. Landing Gear Panel with Knob Down.
The gear knob is shown in the down position and the lights are "lit" with the black text on a green background texture file.

4.5 User Interface

The user interface modifications to the VC includes a mouse activated input interface and an improved Head Mounted Display (HMD) interface. These interfaces, adapted from the AFIT Information Pod, and the hands on throttle and stick (HOTAS) are the input and output suites the pilot will use to interact with the VC. The interface suites will be demonstrated at the 1994 Air Force Association (AFA) Conference in Washington D.C.

Mouse Activated Interface. A mouse activated point-and-click interface installed for the pilot's use allows for a flexible, easily modified method of selecting from the hundreds of switch and knob positions possible on the controls. This is an improvement on the previous interface which, until this year, has been the HOTAS. The HOTAS interface has buttons which the pilot could use to control the sensor suites, the weapons load, and the landing gear. The 1994 VC will retain the previous stick controls in addition to the mouse interface described below.

The mouse activated interface, based on the AFIT Information Pod (Kestermann) interface for the Satellite Modeler (Vanderburgh) and the Synthetic Battle Bridge (Rohrer; Kestermann), is a robust system adaptable to various input methods. The interface establishes invisible planar surfaces, called mouse-panels, within the simulation (Figure 4-16). While a simulation may have many mouse-panels, only one will be the active mouse-panel with a mouse cursor displayed on its surface. Figures 4-13, 4-14, 4-16, and 4-17 show the mouse-cursor on its invisible mouse-panel surface. Additionally, the mouse-panels will have one or more sub-panels. On the sub-panels are the input-activated areas called buttons, visible in Figure 4-14 and Figure 4-19 on the Main Instrument Panel warning lights. Whenever the mouse's left button is 'clicked' and the cursor is on one of the button areas, that button is activated. Clicking the right mouse button activates the simulation's mouse-panels in a cyclical order.

In the VC, the left and right consoles and the main instrumentation panel will each have a mouse-panel just above its surface. The pseudo code for placing a mouse-panel and buttons is shown in Figure 4-17. By co-locating the mouse-panel's button areas with the switches and knobs, the VC's pilot can input commands by placing the mouse cursor over the switch/knob and 'clicking' the left button.

```
Initialize Airplane ()                    // Initialize starting state and sensor displays
  Initialize Instrument Panel             // Place all models on front console
    Place the Static models on                // Static models are static with respect to
      main instrument panels                      the VC, not the virtual environment
    Initialize starting state of dynamic    // Starts the VC in a known state
      models on main instrument panel
    Place dynamic models on main              // Place knobs, switches, and warning
      instrument panel                            lights on main panel
  Initialize Left and Right Panels        // Identical in structure to Initialize Panel
  Initialize mouse-panels (xyz, hpr       // Defines 2D panels, the boundary area of the mouse
          scale, high-light model)
    Initialize sub-panels ()              // Defines position of toggle buttons on mouse-panel
      Set Button Position (length, width      // for each button on panel, set the dimensions
          xyz-position, colors, text)                 & position
  Initialize Mouse                        // Initialize mouse
    Add Panels                                // Pass location of each mouse-panel to
                                                    mouse
  End Initialize routine.
```

Figure 4-17. Pseudo Code for Initializing Cockpit Models and Mouse-Panels.

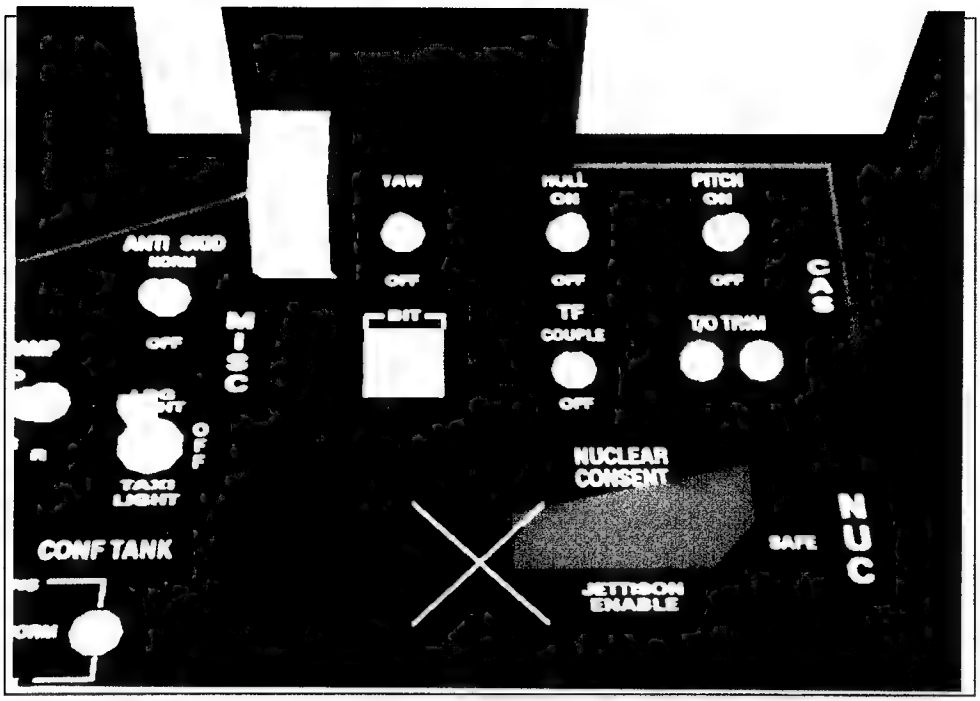


Figure 4-18. Mouse Cursor on the Left Console Nuclear Panel.

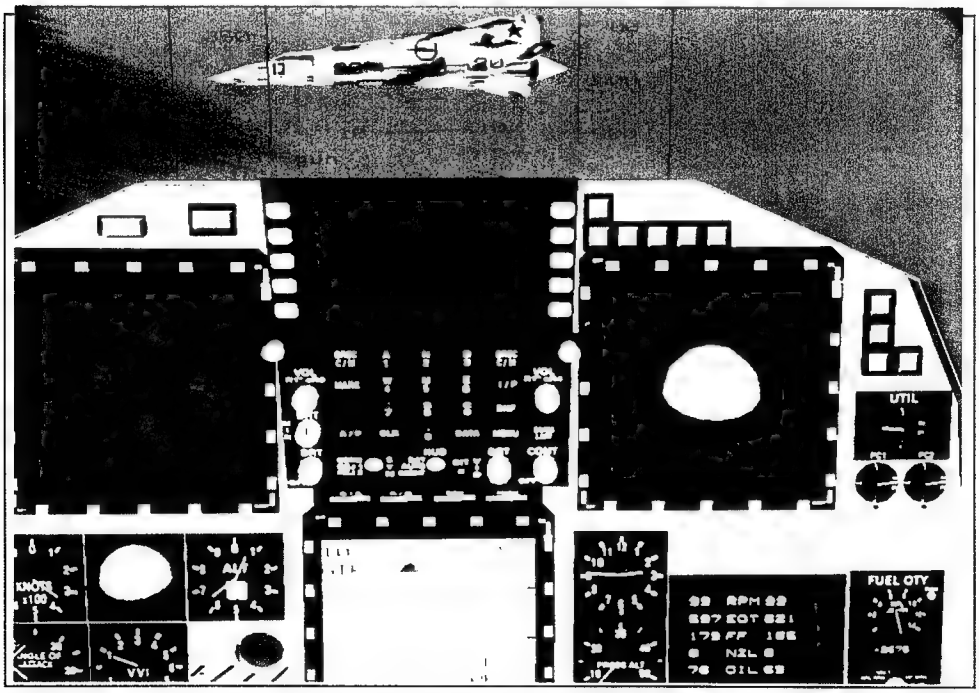


Figure 4-19. Main Instrument Panel with toggle buttons visible over the warning lights. The red outline across the top of the panel indicates this is the active mouse-panel. The mouse is visible in the bottom left hand corner of the photograph.

```
HMD Update Routine
                                        // Update position.
  Read Fastrak (XYZ) Raw-position       // Get raw data from Polhemus.
  VC's (X) =    Fastrak (X) / 50.0      // Adjust scale to VC's virtual environment.
  VC's (Y) =    Fastrak (Y) / 50.0      //
  VC's (Z) = (-1) * Fastrak (Z) / 50.0  // The VR z-axis is pointing down
                                        //    Multiply by -1 to normalize movement.
  View (XYZ) = VC's (XYZ)               // Set view position within VR.
                                       // Update heading(H), pitch (P), and roll (R).
  Read Fastrak (HPR)                    // Get raw data from Polhemus.
  VC's (H) = (-1) * Fastrak (H)         // Multiply raw heading by (-1) to set view
                                        //    away from Polhemus; behind user.
  VC's (P) =    Fastrak (P)             // Pitch and Roll in VC's VR is unchanged
  VC's (R) =    Fastrak (R)             //    from raw data.
                                        //
  View (HPR) = VC's (HPR)               // Set view hpr within VR.
End HMD Update Routine
```

Figure 4-20. HMD Tracking Routine.
This routine tracks from behind the user.

The Head Mounted Display Interface. A new HMD interface replaces the previous HMD. The new design provides a simple-to-use, immersive environment by modifying and replacing most of the functionality of the old HMD interface. Like the old interface, the new HMD will use a Polhemus 3-Space Fastrak mounted at the top of the skull to transmit the view angle and view point to the simulation. Unlike the old interface, however, the new HMD is developed as the primary viewing interface for the VC. In order for the HMD to be an acceptable interface, the display must be smooth (no jitter), adjustable, and easily initiated.

To achieve a smooth HMD view, this year's design tracks from behind the head rather than in front. The new tracking position results in a smoother HMD view of the virtual environment because the Polhemus transmitter can be mounted closer to the receiver from a position behind the user (Figure 4-20 and Figure 4-21).

An adjustable HMD interface is also an important part of the view interface. With the old HMD, if a pilot was too far from the VC's instrument panel, the pilot needed to move forward by adjusting her/his real world position. The new HMD provides the ability to "move" the pilot's seat position within the virtual environment. As a result, the pilot's real world position becomes stable.

The change from movement in the real world to movement within the virtual world makes the interface adaptable to the multitude of users who will fly the VC during the AFA convention. Finally, the new HMD interface is installed at start-up automatically rather than as an option selected at start-up. The simple initiation procedure is designed to encourage use of the HMD interface by removing the start-up options.

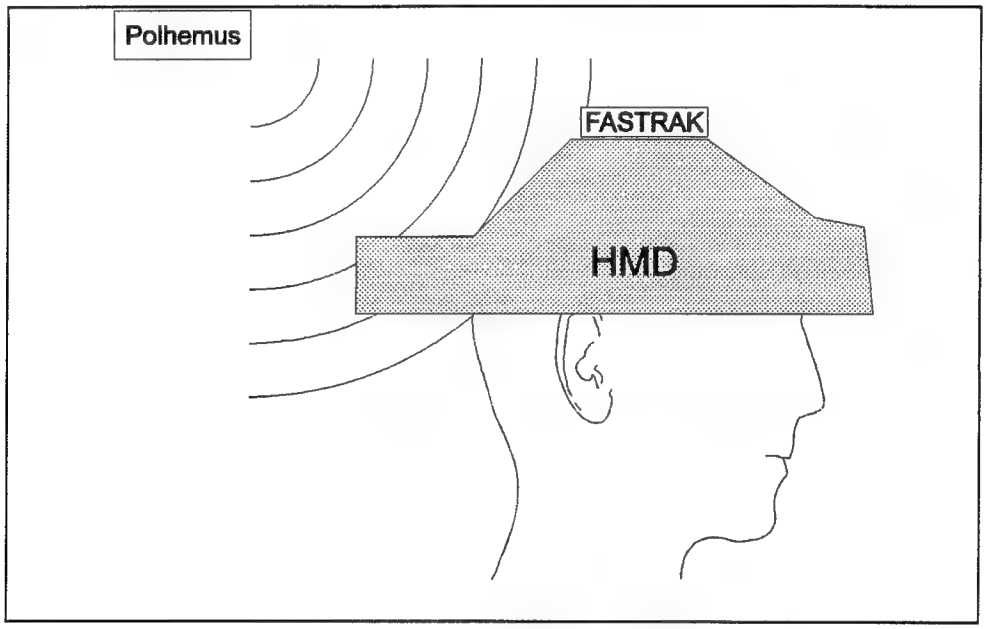


Figure 4-21. The Polhemus and Fastrak.
The Polhemus is located above and behind the user.
The proximity of the Polhemus to the Fastrak helps reduce jitter.

4.6 Conclusion

This chapter described the design and implementation of functioning models and a new interface within the photo-realistic VC. The model construction process produces realistic static controls which can be combined to form aircraft components. The static component models are 'dropped' into the VC. Once in the VC, a standard series of steps is used to replace static models with functioning models which use a DCS matrix. As the models become functional, an immersive HMD view (used for locating and toggling switches) becomes more important.

The VC has two views into the virtual world, the HMD view and the window view. The simulation interface serves to enhance the detail modeled forward crew station. When wearing the

HMD, VC pilots can readily look ahead into a maneuver or check the position of switches and dials within the cockpit. On the other hand, with the VC in window mode, the simulator can be flown like a standard computer simulation. Also, when flying in window mode, switch selection becomes a non-practical series of key strokes and mouse movements difficult to master without practice.

The mouse and keyboard interfaces are designed for use with both HMD and window views. In the window view, the keyboard inputs are used to change the view - turning the virtual head. In HMD mode, the keyboard functions allow the pilot's base view into the virtual world to be altered (similar to adjusting her/his seat). Once the base view point is in place, however, the keyboard is no longer needed.

V. Results and Recommendations

5.1 Overview

This chapter presents the results of the Photo-Realistic AFIT VC including: 1) modeling, 2) mouse interface, 3) button activated switch movement, and 4) HMD interface. In addition, observations from the 1994 Air Force Association Convention (AFA) are delineated. Lastly, recommendations for future research are outlined.

5.2 Results

Modeling. The forward crew station for an F-15E is constructed from schematics, photographs, and sample components available at various organizations at Wright-Patterson AFB. Using standard pieces such as the knobs and toggle switches, the various panels and displays individually before fitting them together to form the consoles and to complete the main instrument panel. The standard pieces are the largest contributors to the model's complexity. Of the 1648 polygons in both the left and right consoles, the knobs and toggle switches contribute about 85% of the total polygonal count. The models and the schematic drawings on which they are based are illustrated in Appendix A. Specific modeling details are presented in this chapter.

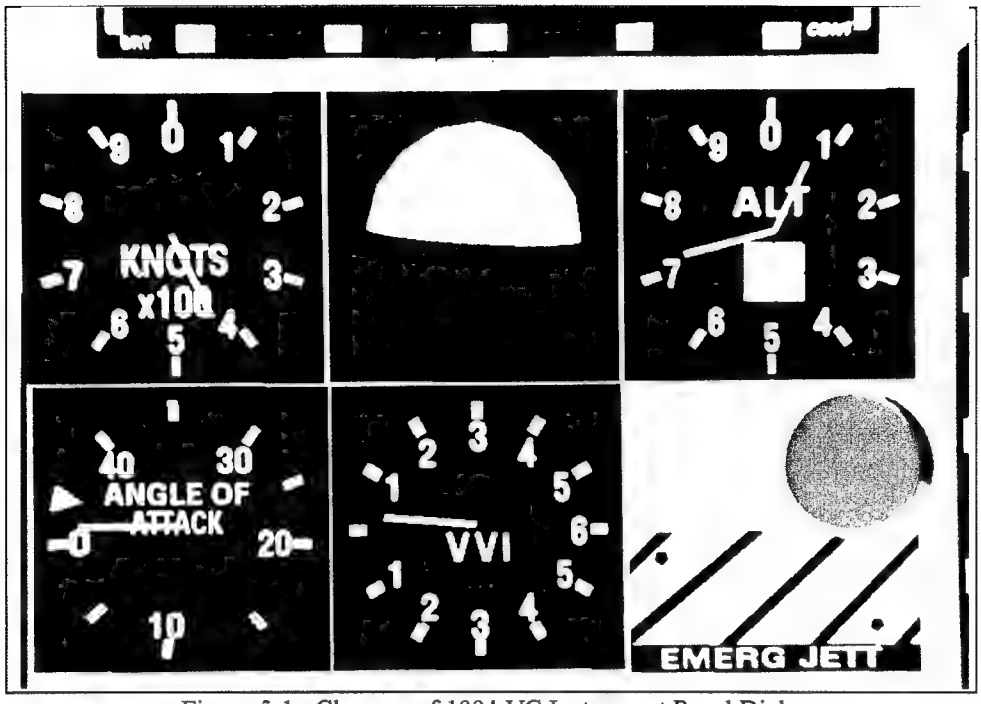


Figure 5-1. Close-up of 1994 VC Instrument Panel Dials.
The text used in these dials was generated by applying texture to polygons.
For example, the bitmap, KNOTS, is on one polygon.

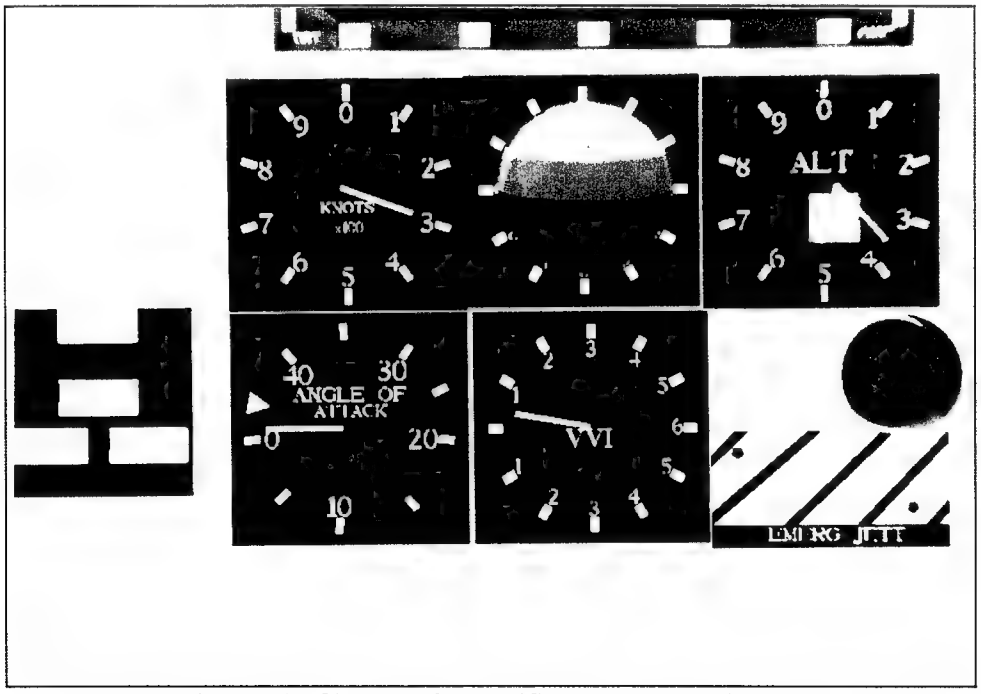


Figure 5-2. Close-up of 1993 VC Instrument Panel Dials.
The text used in these dials was generated with MultiGen's polygons fonts.
Each letter or number consists of multiple polygons.

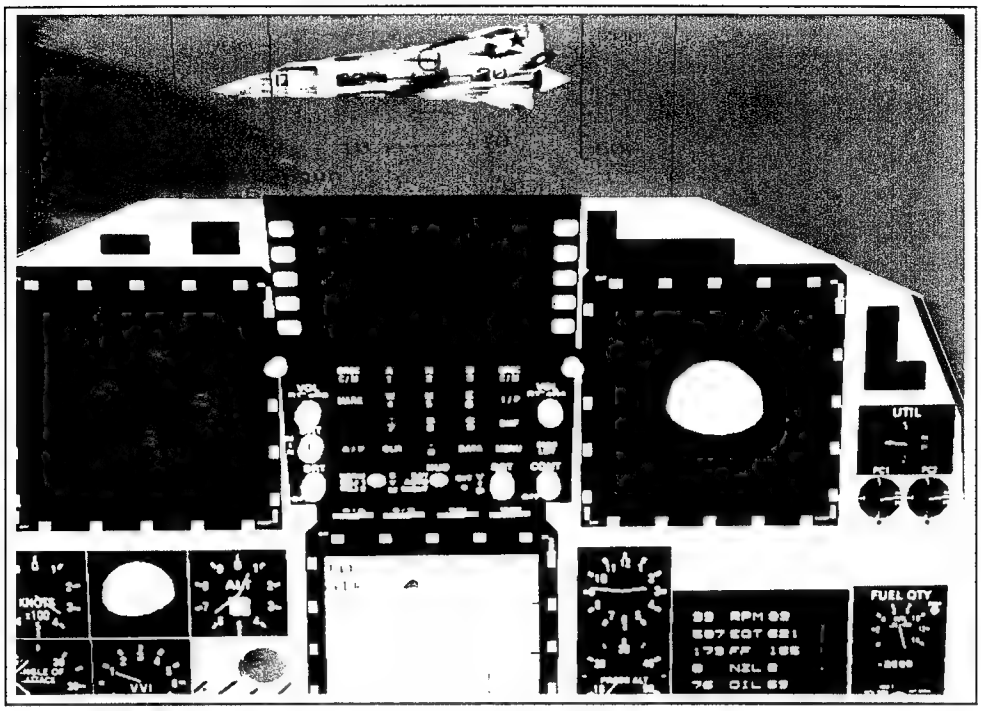


Figure 5-3. View of 1994 VC Instrument Panel.
The texture words and numbers remain distinctly visible.

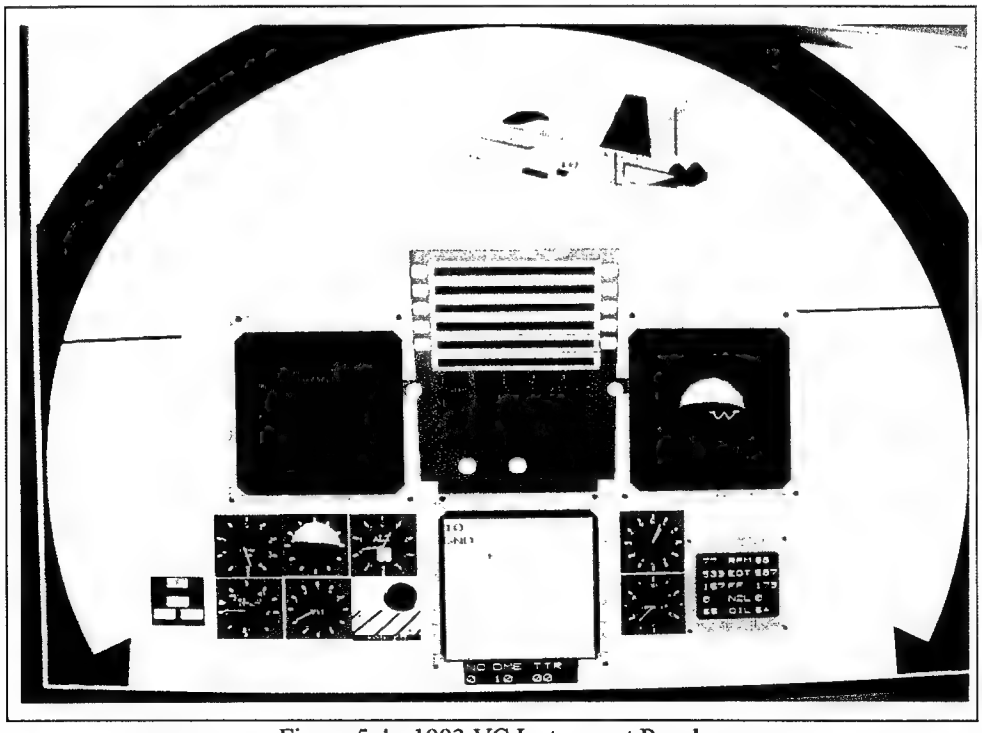


Figure 5-4. 1993 VC Instrument Panel.
The polygonal text in the dials has blended with the background.

The 1994 VC uses texture maps to display most of the words on the main instrument panel and consoles. Except for the dynamic readouts such as the velocity indicator on the Heads Up Display (HUD) and the engine monitor display, all other text in the photo-realistic VC is modeled with bitmaps (Figure 5-1). The bitmaps are more readable within the virtual environment than the labels printed using MultiGen's polygonal text function (Figure 5-2). For example, the bitmapped numbers on the clock in Figure 5-3 remain distinctly visible while the polygonal text in Figure 5-4 appears to vanish into the background. Interestingly, from a close-up view point, the clean lined polygonal text is more readable than the blurred bitmapped text. As the view point moves away, the blurred text retains its general shape and readability while the polygonal text fades away and loses its distinctive shape. This phenomenon is due to the size of the polygons used in each text method. The polygonal letters have polygons fitted together to create a letter. For example, the number "12," visible on the clock dial in Figure 5-4, uses thirteen differently shaped polygons (Figure 5-5). The smallest of these polygons quickly falls below the resolution of the display as

Figure 5-5. Polygonal Text

the view point moves away. Thus, the smaller polygons seem to fade away as shown in Figure 5-4. On the other hand, the textured polygons are as large as the words. The word textures are visible because the polygonal surfaces on which they are written remain larger than the resolution of the display.

Mouse Interface. The mouse interface provides a useful tool for interacting with the VC. The motion of the mouse's cursor is two dimensional and easy to follow along the surfaces in the VC. The mouse is a good first cut at communicating commands into the virtual environment by direct 'handling' of the objects within the environment. However, this interface is not a final solution.

The mouse interface suffers from incompatibility with the HMD and from its head down functionality. The mouse is incompatible with the HMD because the mouse device is not stationary. While wearing the HMD, objects like the control stick and the throttle must remain stationary to be easily found. Because the mouse is designed to move around, a user with an HMD will fumble about time she/he needs to find the mouse. While a trackball can solve this problem, it will not solve the second problem - head down operation. To use the mouse interface, the user must find the cursor, move the cursor over the switch, and toggle the left mouse button. Although these operations can be accomplished quickly, the user must be looking down at the switch she/he is trying to toggle. While looking into the cockpit, the pilot loses all visual motion cues which help him/her to control the aircraft. Even with much practice, I have often looked up from a switch to find the VC in a spin, a steep climb, or a steep dive into the ground.

Button Activated Switch Movement. Switch movement occurs when the corresponding button on the mouse-panel is activated. When a button is activated, a global flag is raised that allows the model control function to modify the switch display.

The switch's range of motion is controlled by a case statement in the appropriate model control function (i.e. models on the main instrument panel are controlled by the instrument.cc). Figure 5-6 is the portion of the code that controls the landing gear knob and its associated lights. Other models can also be controlled from this point if needed.

HMD Interface. The HMD interface greatly adds to the utility of the detailed three-dimensional models. As the pilot wearing the HMD turns, the relative positions of the models change due to parallax, an effect which enhances the perception of three dimensions (Zeltzer and Drucker).

Another factor influencing the utility of the HMD is the quality of the display. The AFIT Graphics Lab has two HMDs which is employed with the VC during the year. The first HMD, a PT-O1 with a 420 * 230 display (OPTICS 1), is an inexpensive and rugged device which is used for test and development. For interaction with the virtual environment, however, the PT-O1 is inadequate; the device's pixel and color resolution severely limits the fine details in the environment. The other HMD, an n-Vision HMD with 1280 * 960 pixel format and fifty degree field of view in monoscopic display mode (Lewis), works exceptionally well with the VC environment. The color and pixel resolution matches or outperforms the CRT displays in the Graphics Lab. The visual improvement the n-Vision HMD offers is an excellent tool for demonstrating the capabilities of the fully immersive.

```
// GEAR_KNOB
  if (Globals->Gear_Knob)                          // Gear_Knob is true if the button is pressed
  {
   static float Gear_Knob_Adjustment = 0.0f;       // Variable remembers state of  knob
        float Gear_Lt_Adjustment = 0.0f;           // Default position of gear lights is off
   Globals->Gear_Knob = FALSE;                             // Gear_Knob button must be pressed
                                                   //   again to accomplish this code
  if (Gear_Knob_Adjustment == 0.0f) {              // If the gear knob is up...
      Gear_Knob_Adjustment = 50.0f;                //     put the gear knob model down
      Gear_Lt_Adjustment = 180.0f;                 //     and flip lights to 'on' position
    }
   else                                            // Else
      Gear_Knob_Adjustment = 0.0f;                 //     raise the landing gear knob model
   pfDCSRot(gear_knob->RotDCS, 0.0f, 0.0, Gear_Knob_Adjustment);        // Carry out commands
   pfDCSRot(gear_nose_lt->RotDCS, 0.0f, 0.0, Gear_Lt_Adjustment);       //  and adjust lights
   pfDCSRot(gear_left_lt->RotDCS, 0.0f, 0.0, Gear_Lt_Adjustment);
   pfDCSRot(gear_right_lt->RotDCS, 0.0f, 0.0, Gear_Lt_Adjustment);
  }                                                // Other models can be adjusted here
```

VC. Figure 5-6. The case statement for controlling the depiction of landing gear.

The differences between the HMD and CRT (window) are the views each provides into the virtual environment. The HMD accommodates freedom of motion in the virtual reality while the window view is free from restrictions in the real world. During flight, the freedom of movement the HMD allows does not initially make controlling the aircraft easier than piloting with the CRT view. During the VC demonstration at the AFA Convention the difficulties involved with using an HMD became apparent.

The 1994 Air Force Association (AFA) Convention. The 1994 AFA Convention was the best opportunity this year to have actual pilots, video gaming would-be pilots, and non-computer literate laymen fly the Photo-Realistic AFIT VC. In a three day period, approximately seventy people used the VC, the Synthetic Battle Bridge, and the Satellite Modeler. At the convention, a controller guided the VC simulation with the keyboard functions as the convention attendees flew the VC. The advantages and disadvantages of the HMD were deduced from observing the conference participants.

Two advantages the HMD provided were: 1) the potential for more realistic flight and 2) the capability to remove distractions from the user's view. First, the HMD gave users the opportunity to employ more realistic flight techniques. One experienced pilot used the look-before-you-leap capability to successfully perform 'text book' maneuvers. Second, the HMD presented a more immersive environment with fewer distractions compared to the CRT. At the convention, the immersion phenomenon was demonstrated most often when participants tried to reach for the virtual controls they saw. In addition, participants wearing an HMD were able to fly without distractions. Often, participants were very surprised by the size of the crowd watching their flight because the n-Vision HMD used blocked out the outside world from the participants' field-of-view.

Initially, the HMD was more difficult to use than the CRT because users were unaccustomed to the free movement it offered. While the HMD view depended on the position of the user's head, the CRT's window view remained fixed. For example, conference participants would often 'forget' where the front of the aircraft was pointing. The participants would look towards the point they wanted to fly while the aircraft was headed in a different direction. The

CRT does not have this problem because the view was fixed. People without flying experience (actual or video) had the most difficulty adjusting to the HMD. After watching people make this mistake repeatedly on the first day of the conference, they were instructed on the basics of HMD assisted flight. With these three simple instructions, attendees were noted to be far more proficient flying on the last two days of the conference than they had been on the first day. The instructions and rationale follows.

The participant was instructed to:

1. look toward the front of the aircraft.
   Reason: this act oriented their head to the forward direction.
2. align his/her body with his/her head to face the same direction.
   Reason: otherwise, during the flight, the participant's head aligned with their body to face out the side of the aircraft instead of the front.
3. always keep a portion of the main instrument panel in his/her field-of-view while flying.
   Reason: this provided another point of reference needed for level flight.

Without sensory feedback providing the motion cues of an actual aircraft, participants on the first day of the conference often lost their bearings while flying the VC. These participants would fly roller-coaster flight paths, never actually bringing the nose of the VC and their field of view along the same axis. After receiving pre-flight instructions, most participants did not encounter the same problem. Thus, while not having motion feedback is a problem, people were able to overcome this obstacle once they knew how to properly deal with it.

5.3 Recommendations

The Photo-Realistic AFIT Cockpit has put into place the tools needed to interact with realistic controls within a virtual environment. Suggestions for further research include areas such as improving the: 1) crew station, 2) head-tracking algorithm, 3) mouse interface, 4)structural integrity of the VC code, and 5) transmission of the cockpit's internal status.

First, as a vital component of the immersion phenomenon, the crew station requires much work before it can be considered a fully functional photo-realistic environment. Studies should be

continued on the following aspects: 1) complete the modeling of sub-consoles in the forward crew station, 2) model the rear crew station, and 3) add functionality to the switches in both crew stations. Presently, toggling a switch has no effect on the aircraft other than moving the switch.

Second, the head-tracking algorithm is an area which needs further investigation because the present method is inadequate. The technique for head-tracking converts the raw XYZ coordinates provided by a single Polhemus 3-Space Fastrak into the view's heading, pitch, and roll (hpr) within the virtual environment. The current system generates unnatural motion based on the relative locations of the Polhemus transmitter to the Fastrak receiver. For example, looking down (moving your chin to your chest) while wearing the HMD generates a virtual image akin to placing your head between your knees. With some practice, a user is able to avoid movements which lead to unnatural virtual environment responses. However, requiring users to 'avoid' natural movements is a restriction which dampens the feel of virtual reality. Furthermore, teaching users to avoid actions is not the purpose of the VC. Ideally, natural motion should be reflected inch-for-inch and degree-for-degree in the virtual environment.

Third, alternatives or improvements to the mouse interface should be explored. The current mouse interface requires the pilot to look at the cursor to align it with the switch he/she wants to toggle. Although the actions are easy to master, the amount of time needed to accomplish a simple task is much longer than it would in real life. Other methods which require less look down time should be investigated. For example, new speech recognition systems with large vocabularies may be versatile enough to meet the demands of the VC (Roe). A possible improvement is to allow the pilot to script a planned sequence of cursor movements that she/he will need during a high stress maneuver.

Fourth, research over the past three years has stressed the structural integrity of the code. Students have followed the incremental approach to advancing the VC; each new student built on the work of the previous students. While each student has attempted to maintain a modular approach, every new module added has been slightly different from the previous work in style and function. As a result, the VC code has grown overly complicated. The VC uses global variables

like patch wires to allow communication between modules. I strongly believe that a new truly modular code could be developed using sound software engineering practices. Modularity should then be demonstrated by applying multiple aircraft beds to the VC.

One last area for future research is transmission of the cockpit's internal status to other users on the network with a need to know the status. For example, if a rear station is installed in the F-15E, then an interactive protocol will have to be developed so that the pilot and the backseater may interact within the simulation. Also, such a protocol could send information to an instructor/observer at a remote site. Ideally, the instructor/observer at a single workstation will be able to monitor the progress of multiple VCs.

5.4 Summary

This thesis improves the immersive quality of the AFIT VC by putting the VC's pilot in a virtual environment with the F-15E forward crew station controls and a command interface. The HMD interface has also been enhanced to increase the utility of the new controls within the virtual environment. This research provides the ability to display the status of the controls within the cockpit and to input commands directly into the virtual environment. Future research can advance this work a number of directions such as transmitting and receiving cockpit status to remote F-15E back seat on the DIS network.

Appendix A

Appendix A contains a photograph of the models built for the forward crew station. The models developed for the main instrument panel are shown in A.1 followed by the left console in A.2 and the right console in A.3.

A.1 Main Instrument Panel Models

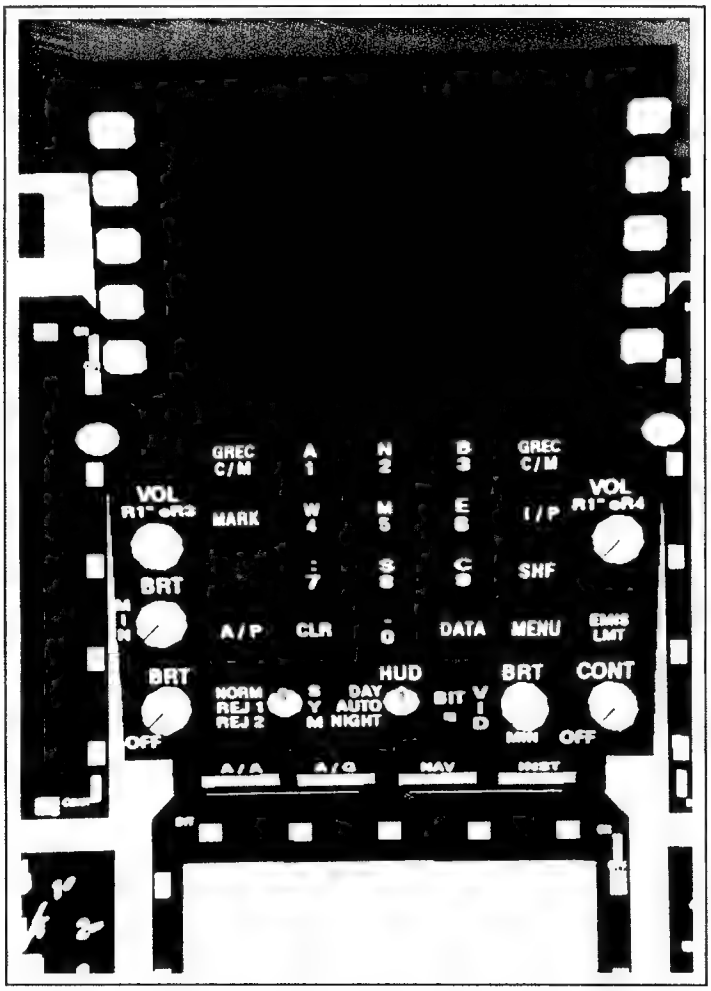


Figure A.1-1. Navigation Panel Model and HUD Display Control Panel.

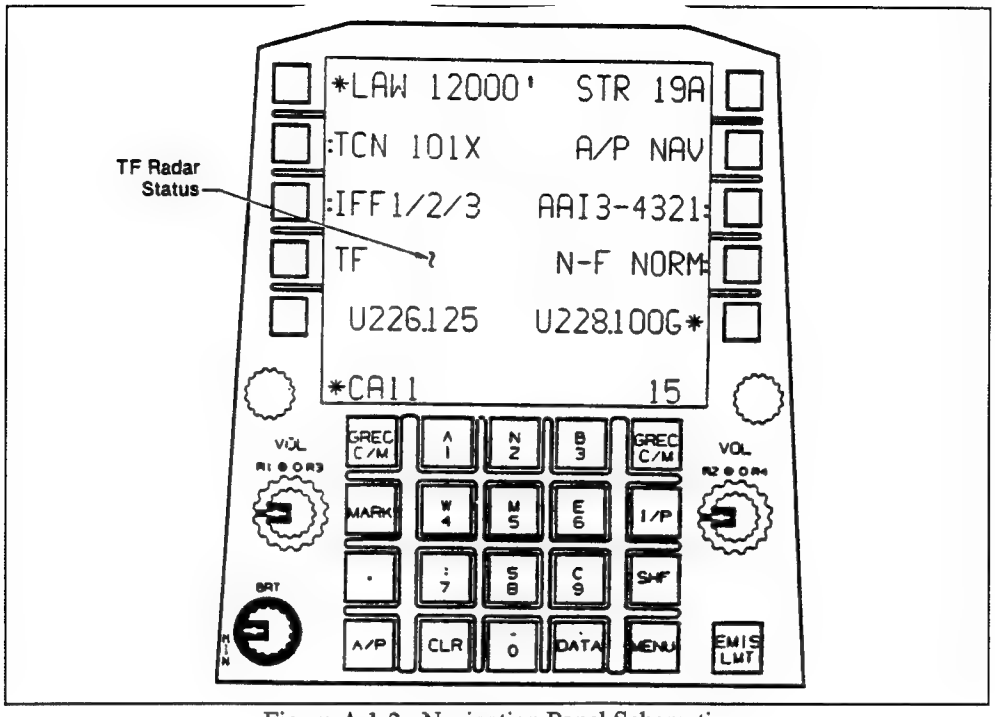


Figure A.1-2. Navigation Panel Schematic.

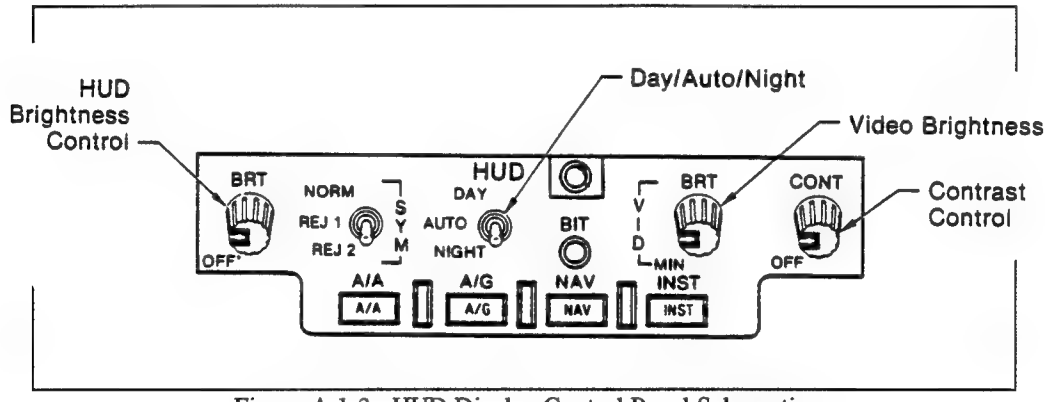


Figure A.1-3. HUD Display Control Panel Schematic.

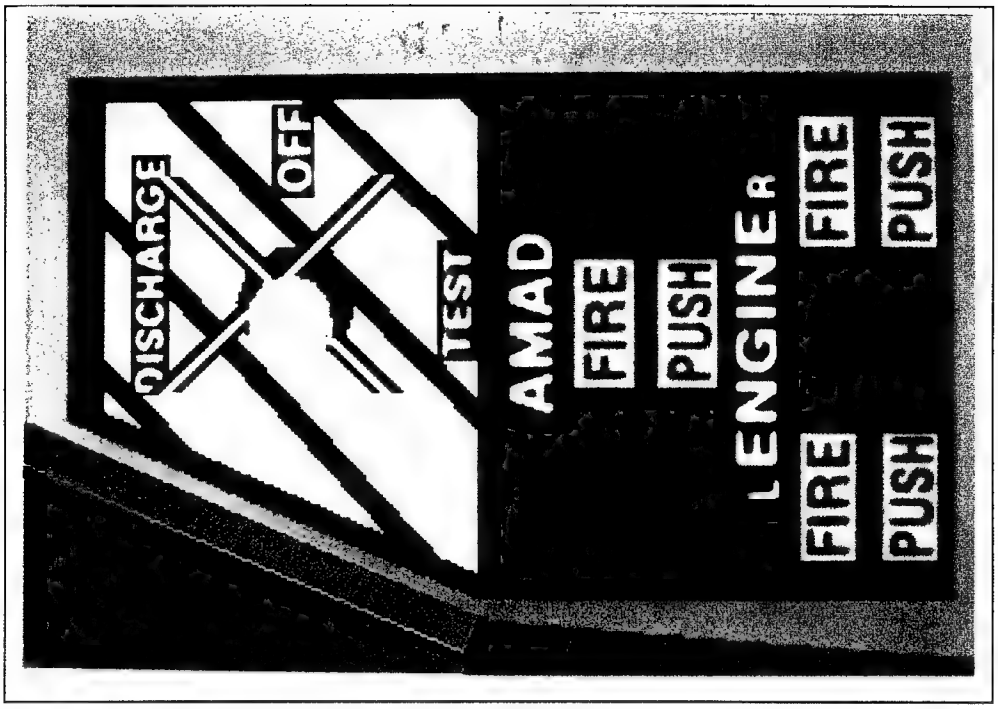


Figure A.1-4. Fire Warning/Extinguishing Control Panel Model.

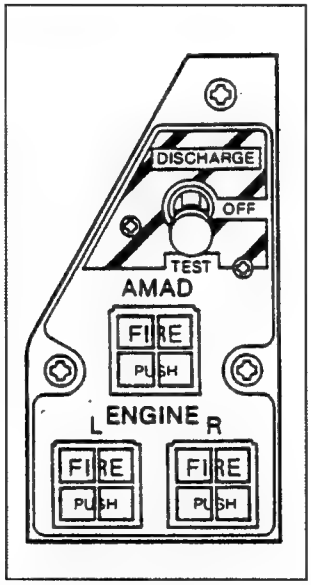


Figure A.1-5. Fire Warning/Extinguishing Control Panel Schematic.

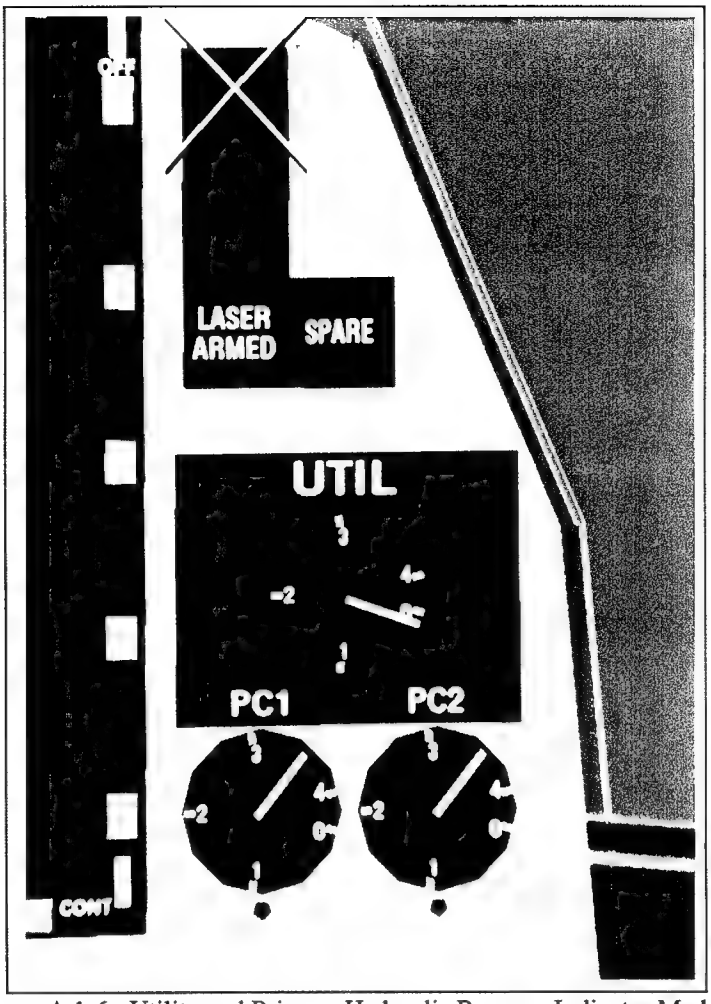


Figure A.1-6. Utility and Primary Hydraulic Pressure Indicator Models.

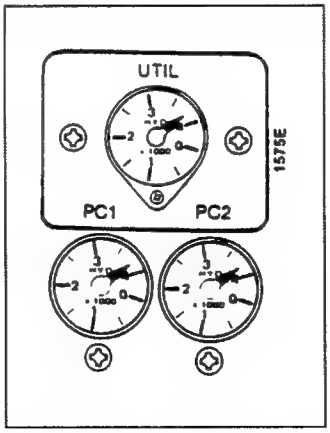


Figure A.1-7. Utility and Primary Hydraulic Pressure Indicator Schematics.

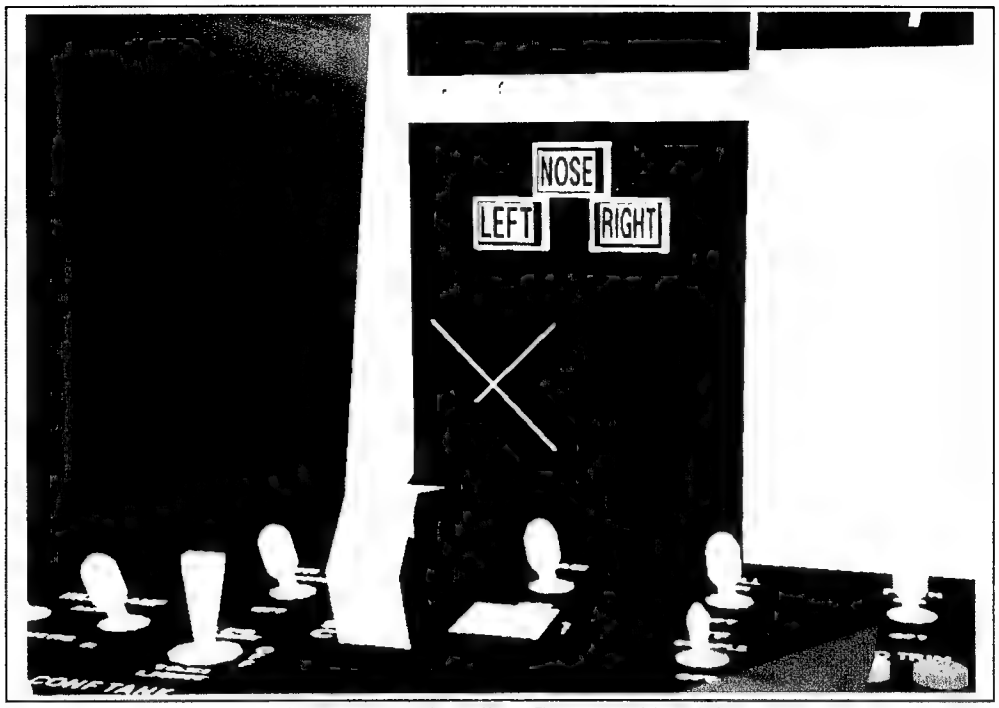


Figure A.1-8. Landing Gear Panel Model.

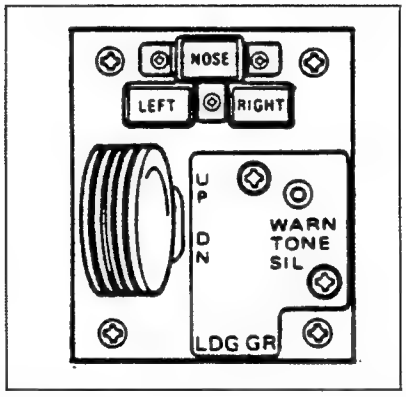


Figure A.1-9. Landing Gear Panel Schematic.

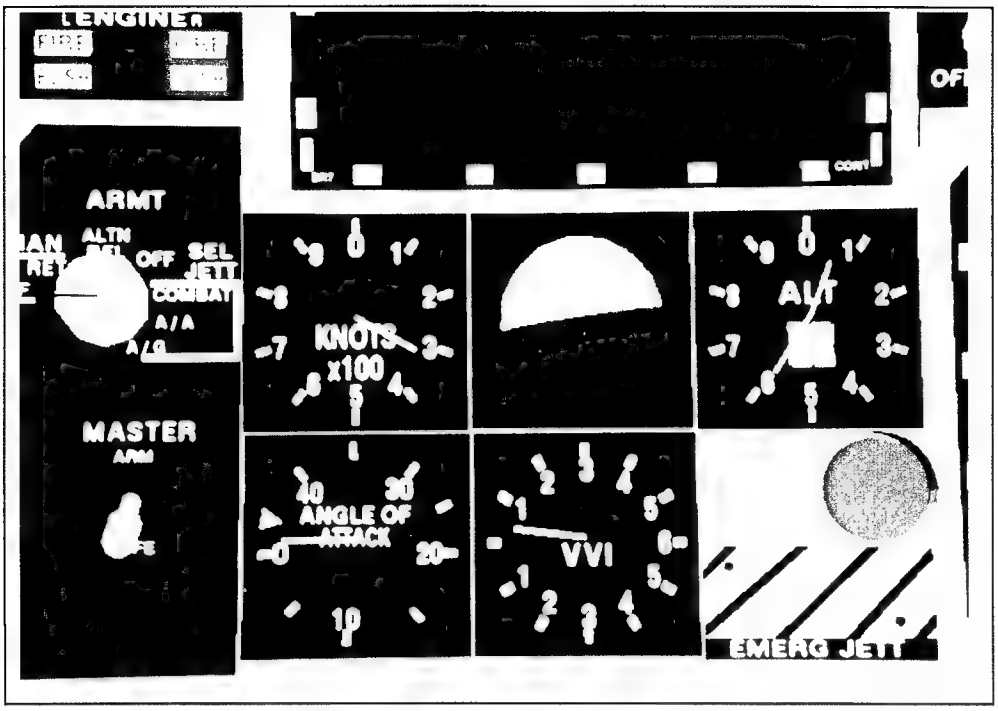


Figure A.1-10. From Left Clockwise: Armament Control Panel, Standby Air Speed Indicator, Standby Altitude Indicator, Standby Altimeter, Emergency Jettison Select Switch, Vertical Velocity Indicator, and Angle of Attack Indicator Models.

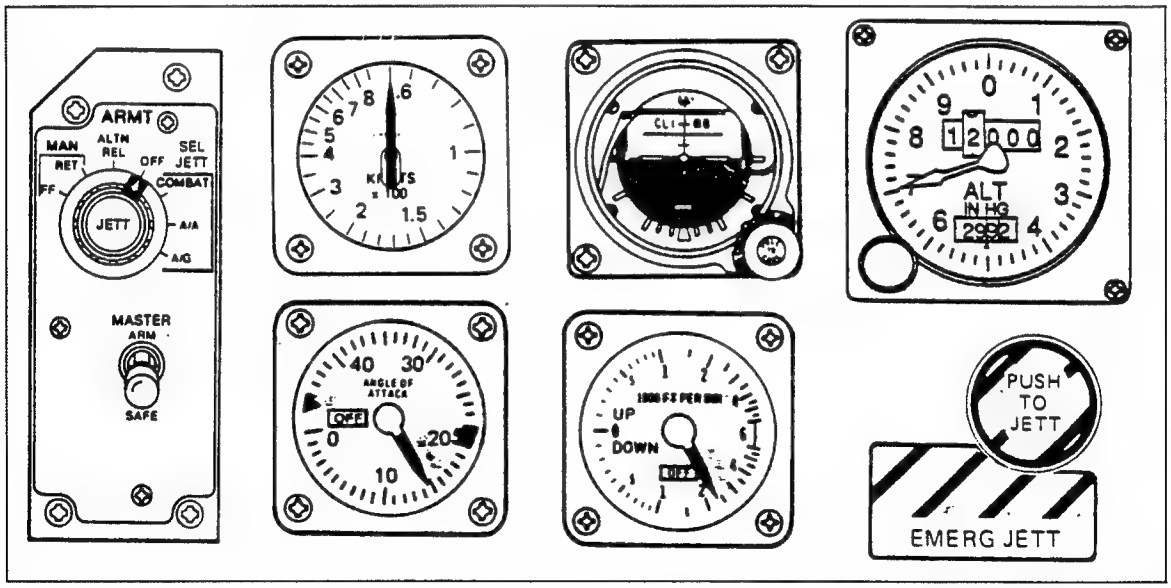


Figure A.1-11. From Left Clockwise: Armament Control Panel, Standby Air Speed Indicator, Standby Altitude Indicator, Standby Altimeter, Emergency Jettison Select Switch, Vertical Velocity Indicator, and Angle of Attack Indicator Schematics.

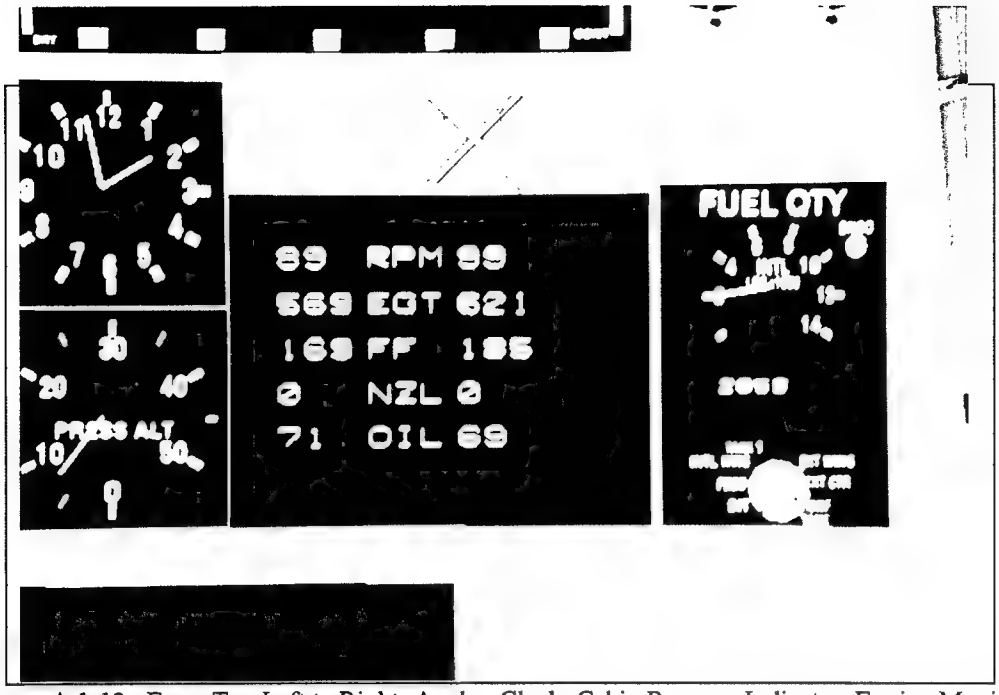


Figure A.1-12. From Top Left to Right: Analog Clock, Cabin Pressure Indicator, Engine Monitor Display, and Fuel Quantity Indicator Models.

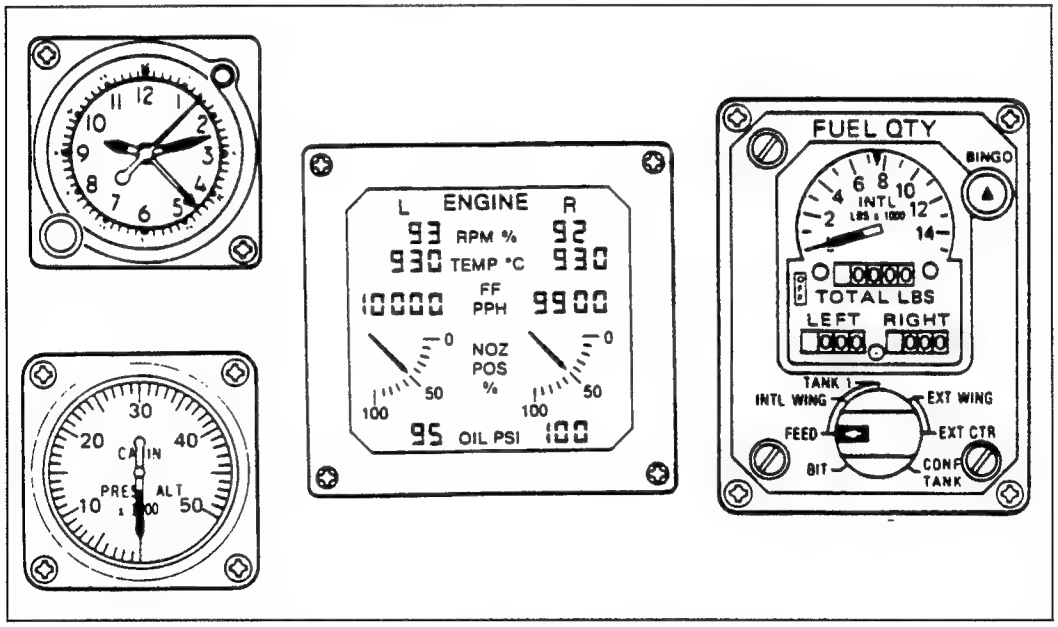


Figure A.1-13. From Top Left to Right: Analog Clock, Cabin Pressure Indicator, Engine Monitor Display, and Fuel Quantity Indicator Schematics.

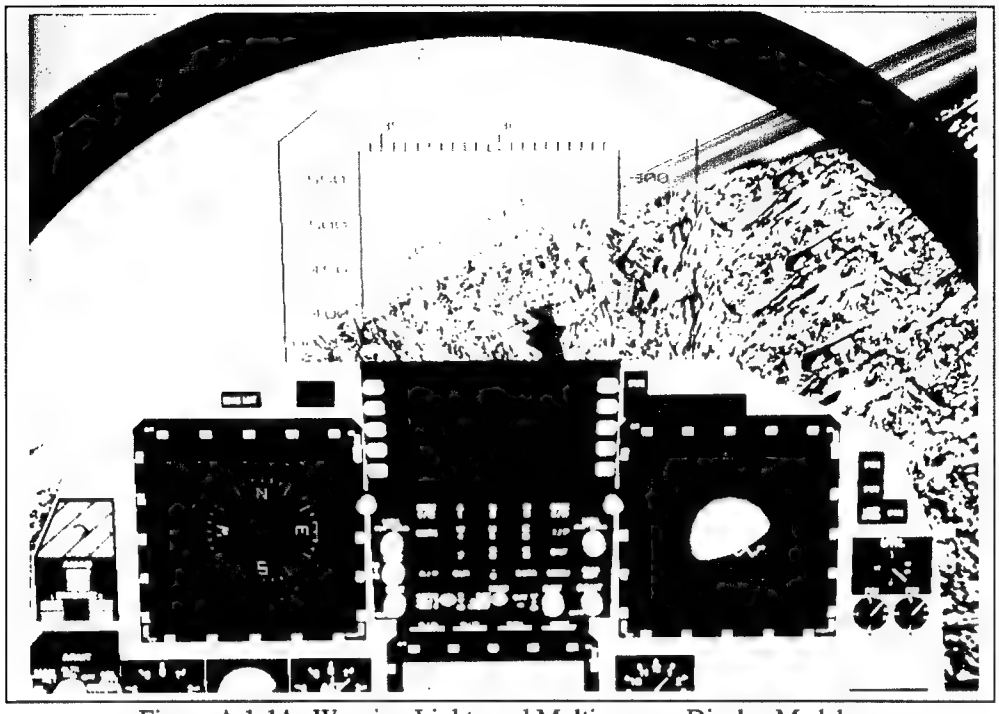


Figure A.1-14. Warning Lights and Multipurpose Display Models.

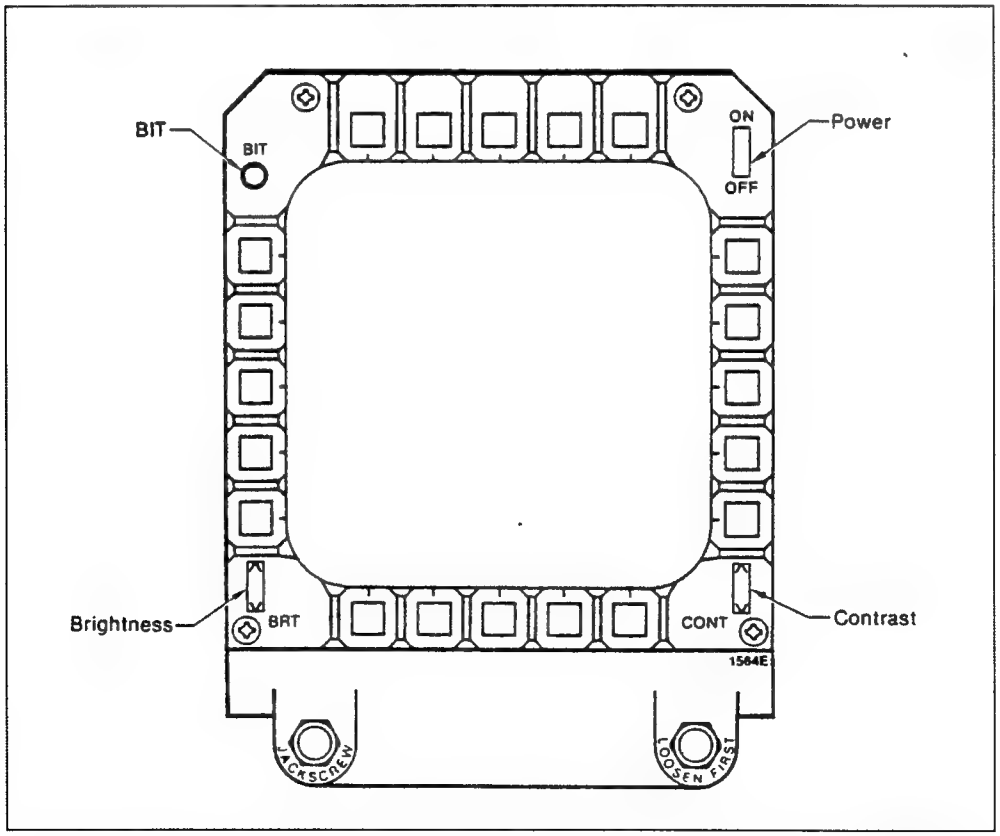


Figure A.1-15. Multipurpose Display Schematic (Warning Lights not Available).

A.2 Left Console Models.

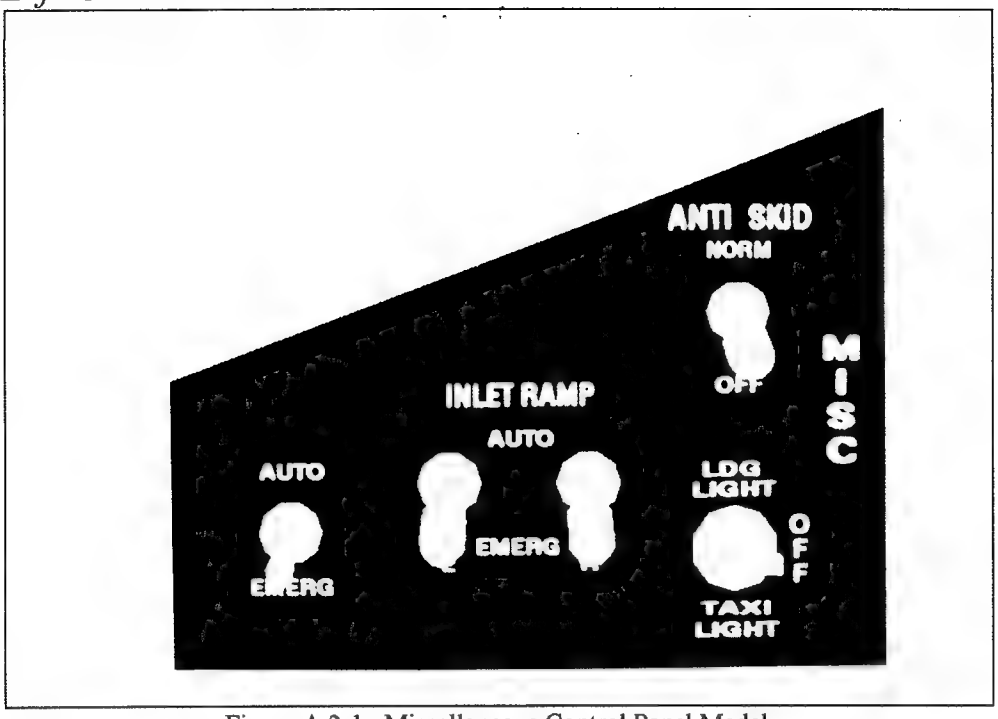


Figure A.2-1. Miscellaneous Control Panel Model.

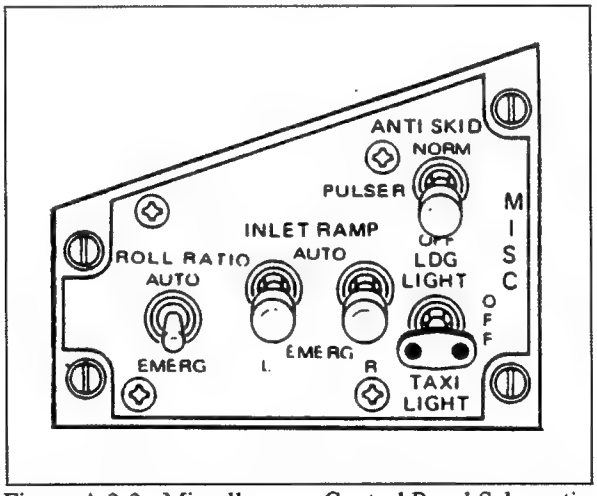


Figure A.2-2. Miscellaneous Control Panel Schematic.

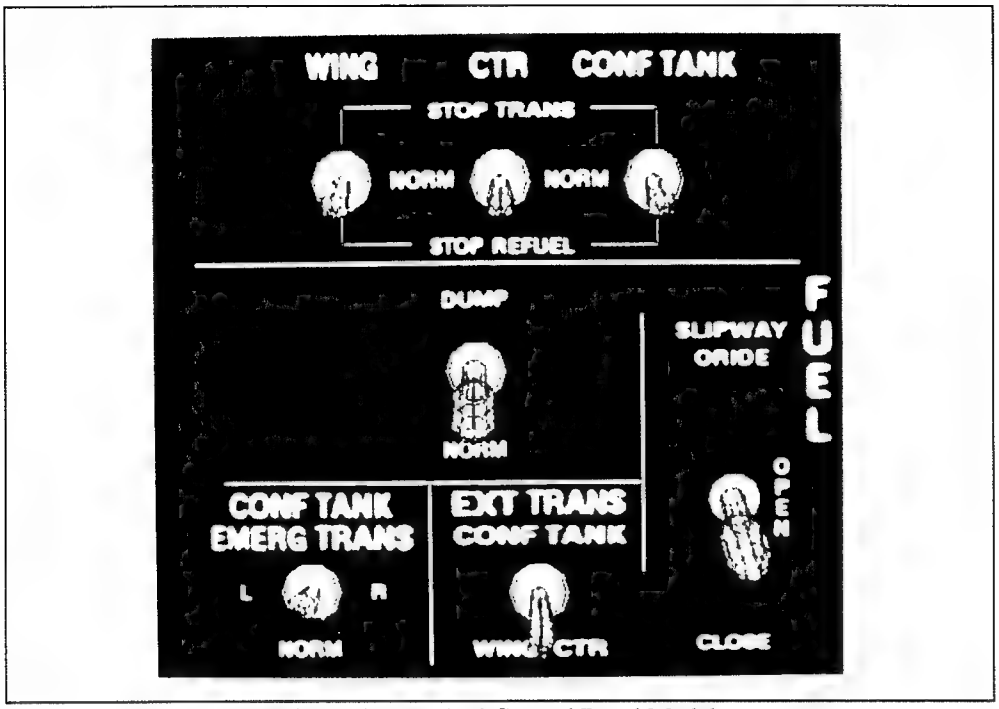


Figure A.2-5. Fuel Control Panel Model.

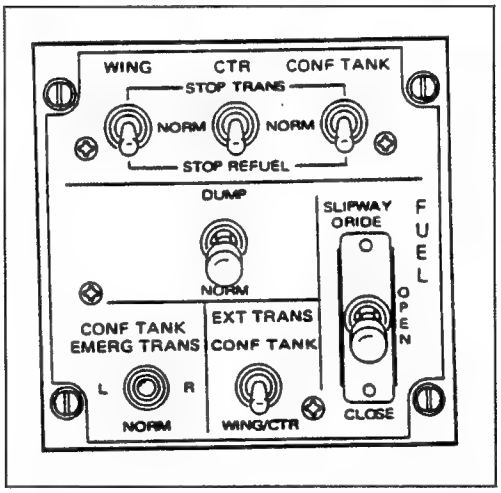


Figure A.2-6. Fuel Control Panel Schematic.

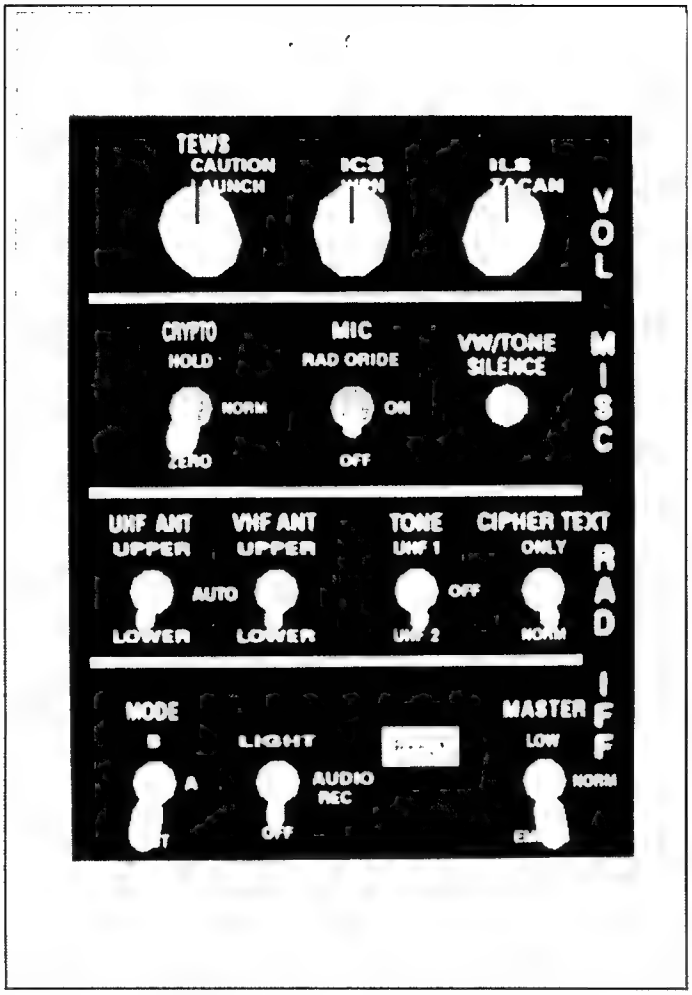


Figure A.2-7. Identification Friend or Foe and Miscellaneous Panel Model.

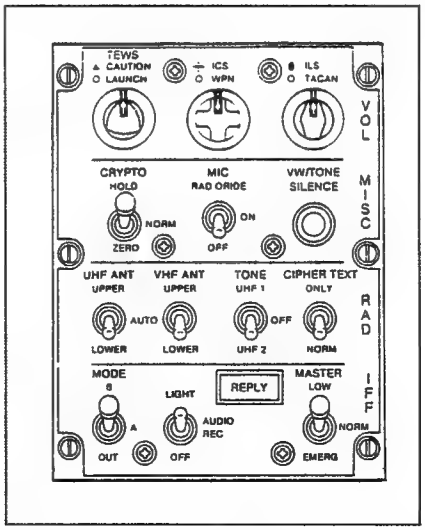


Figure A.2-8. Identification Friend or Foe and Miscellaneous Panel Schematic.

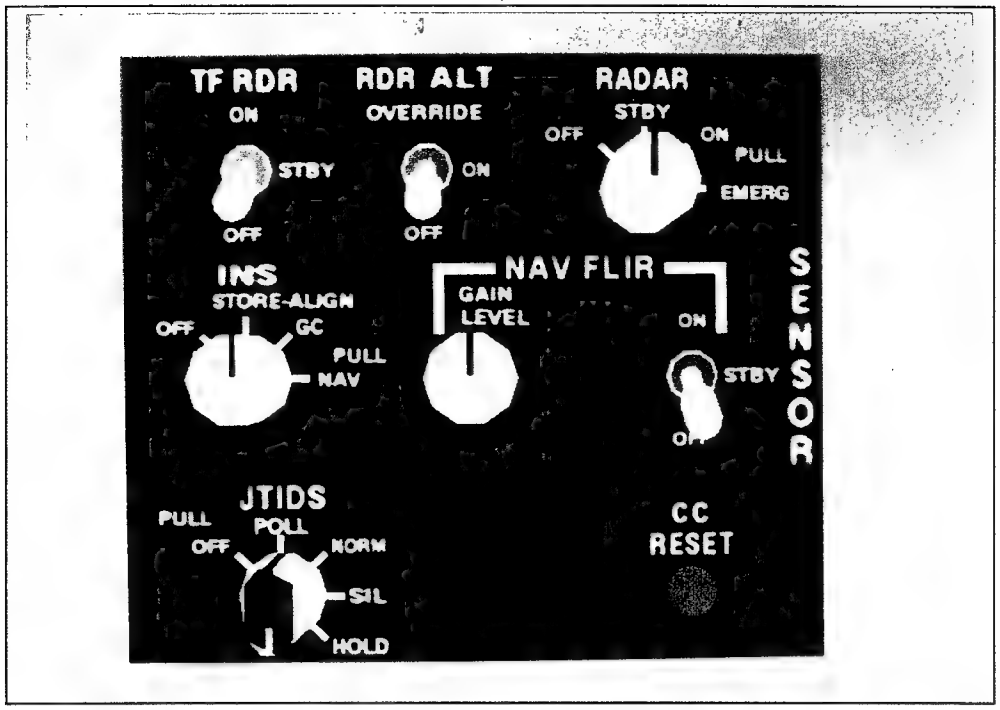


Figure A.2-9. Sensor Control Panel Model.

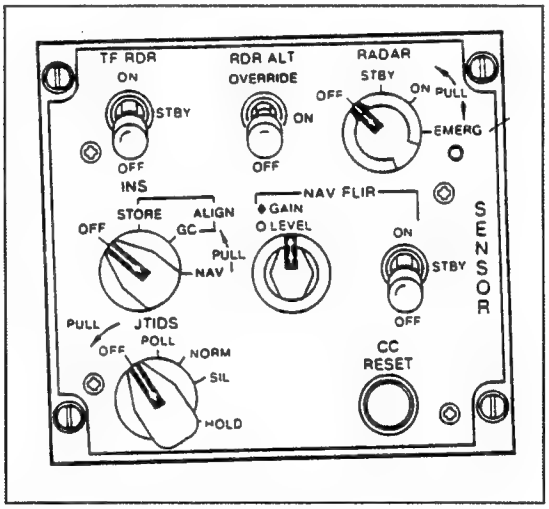


Figure A.2-10. Sensor Control Panel Schematic.

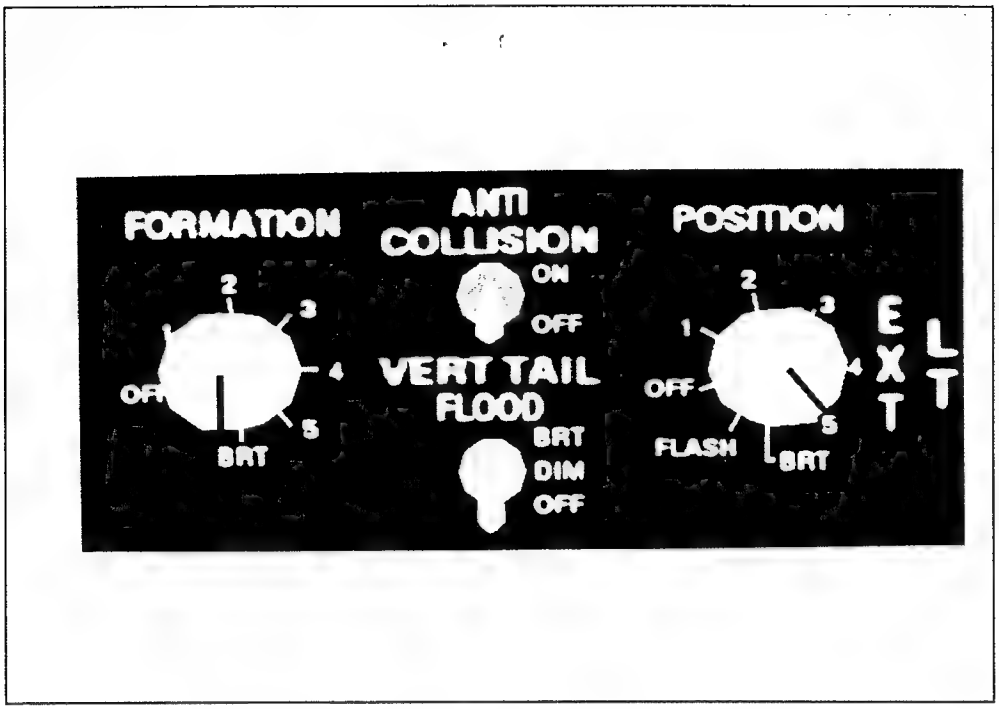


Figure A.2-11. Anti-Collision Control Panel Model.

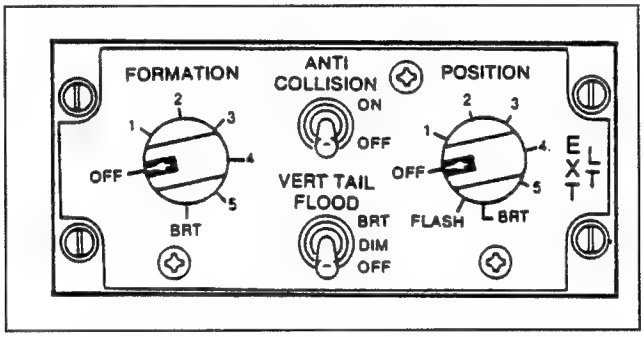


Figure A.2-12. Anti-Collision Control Panel Schematic.

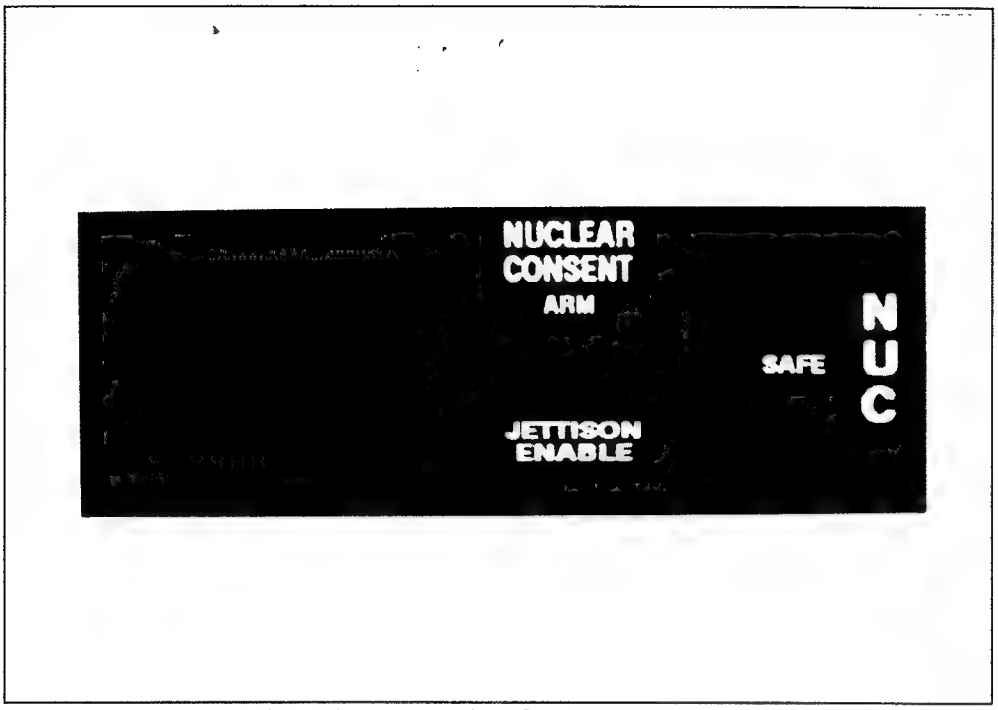


Figure A.2-13. Nuclear Control Panel Model.

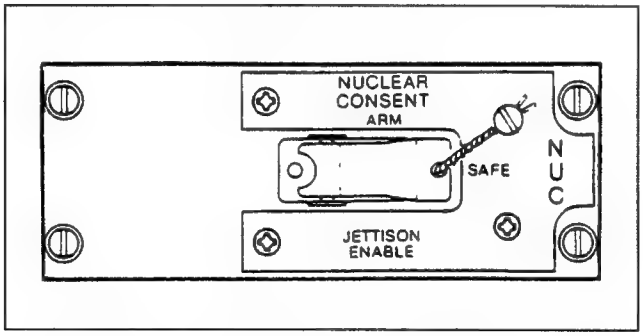


Figure A.2-14. Nuclear Control Panel Schematic.

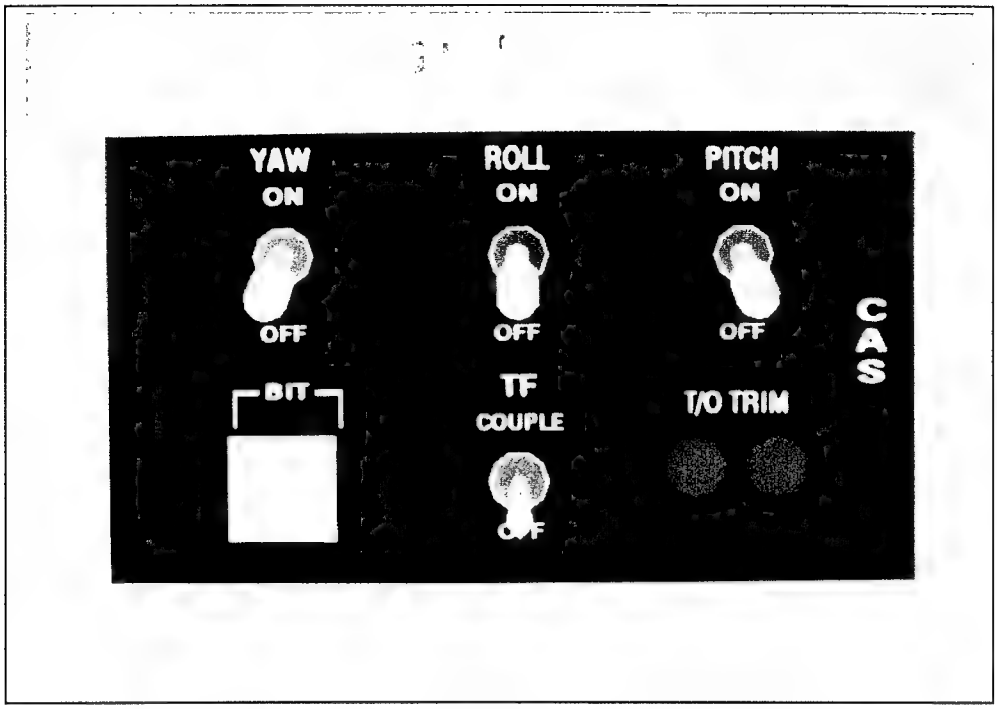


Figure A.2-15. Control Augmentation System Control Panel Model.

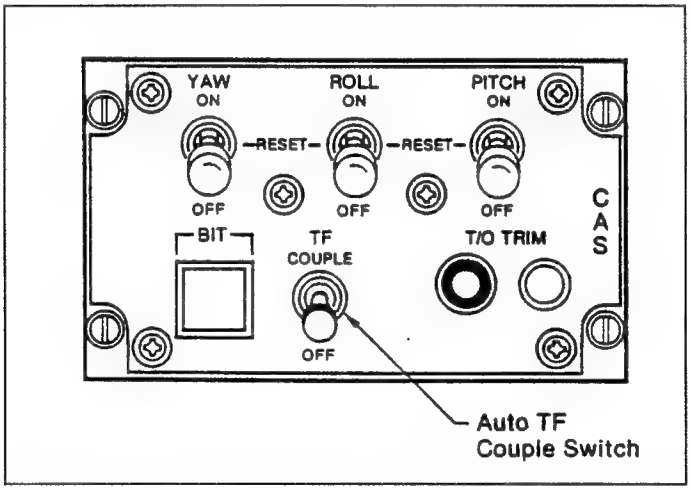


Figure A.2-16. Control Augmentation System Control Panel Schematic.

A.3 Right Console.

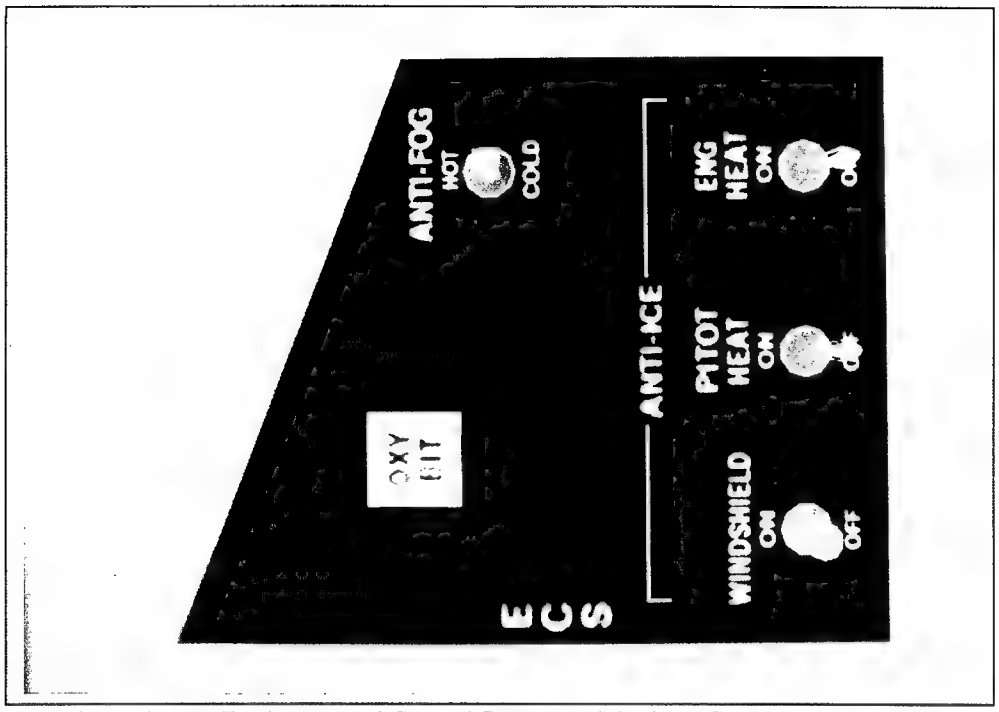


Figure A.3-1. Environmental Control System and Anti-Ice Control Panel Model.

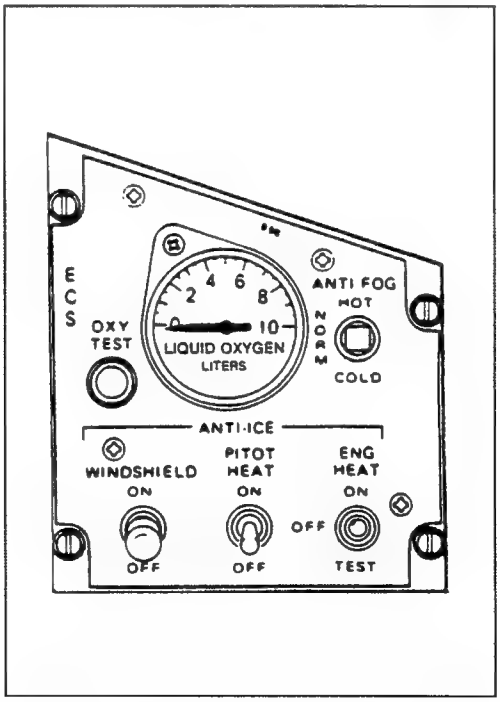


Figure A.3-2. Environmental Control System and Anti-Ice Control Panel Schematic.

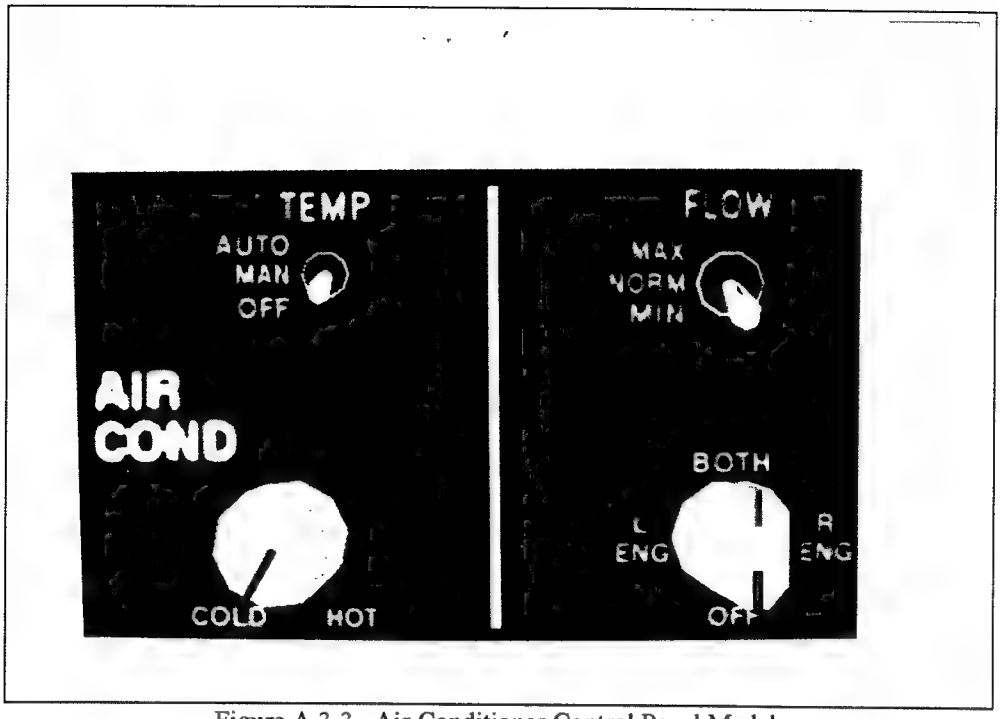


Figure A.3-3. Air Conditioner Control Panel Model.

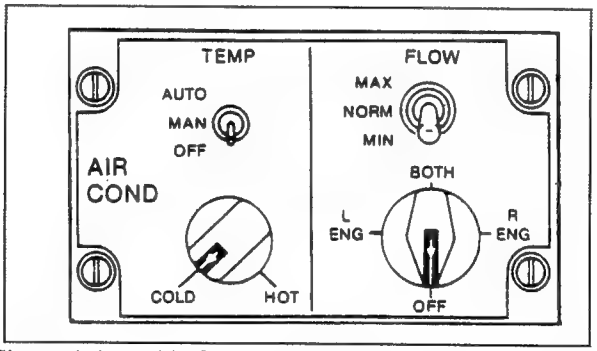


Figure A.3-4. Air Conditioner Control Panel Schematic.

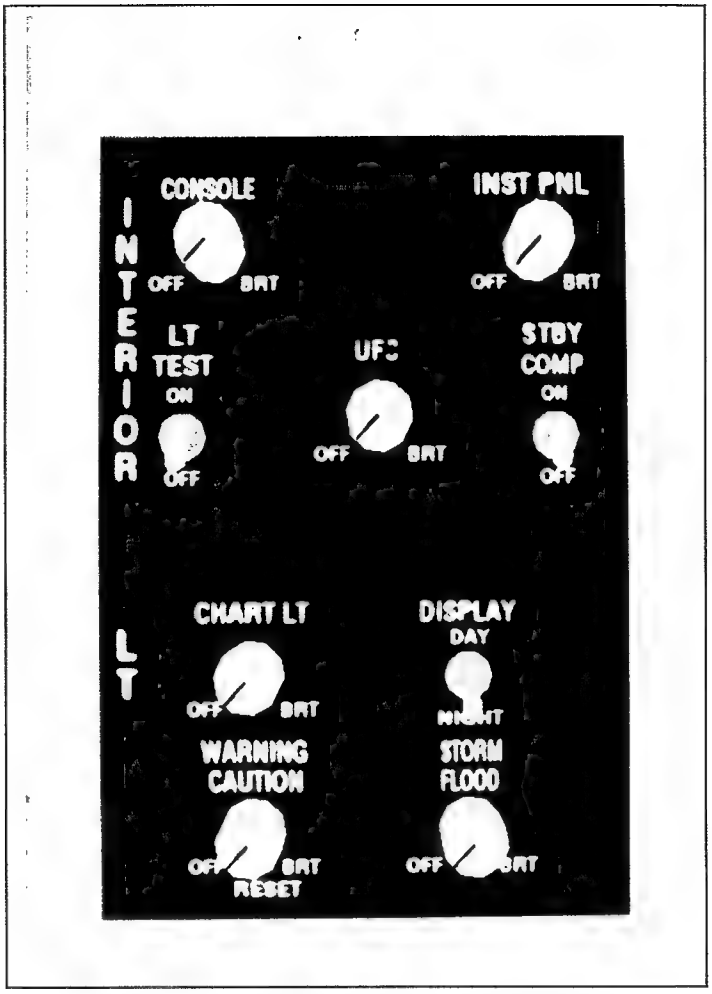


Figure A.3-5. Interior Light Control Panel Model.

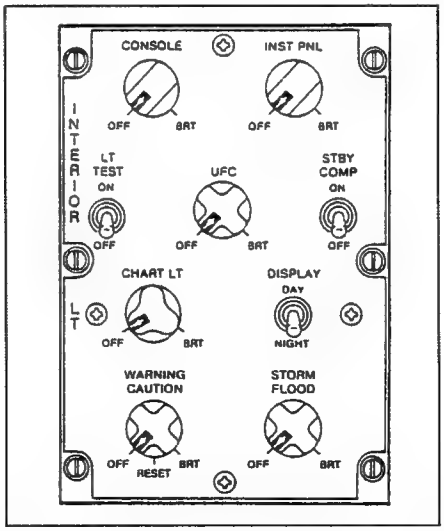


Figure A.3-6. Interior Light Control Panel Schematic.

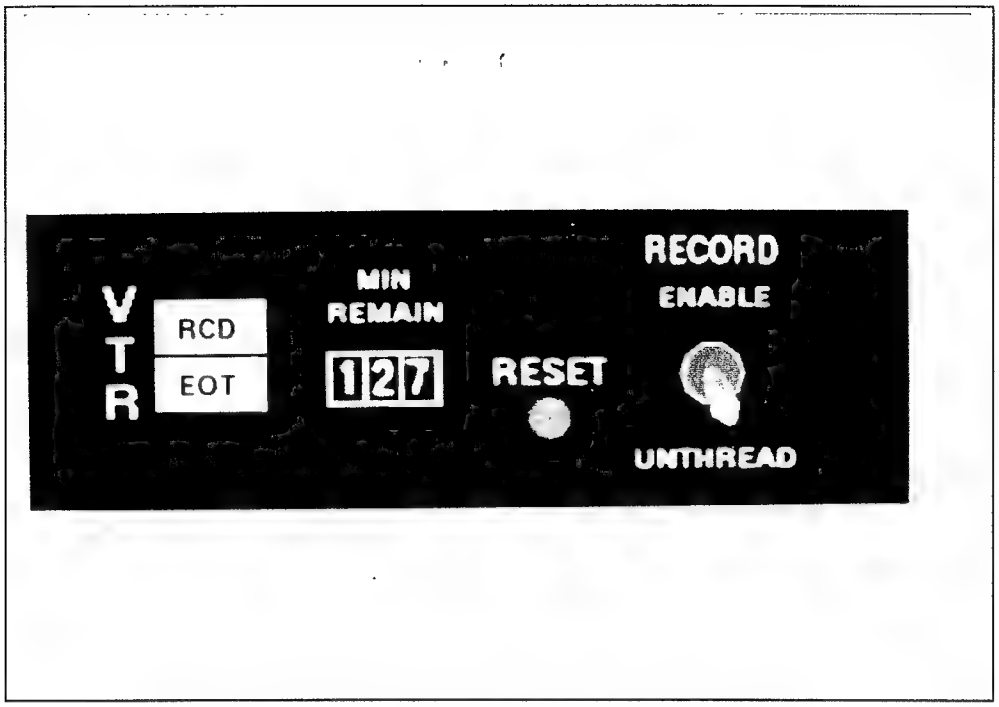


Figure A.3-7. Video Tape Recorder Control Panel Model.

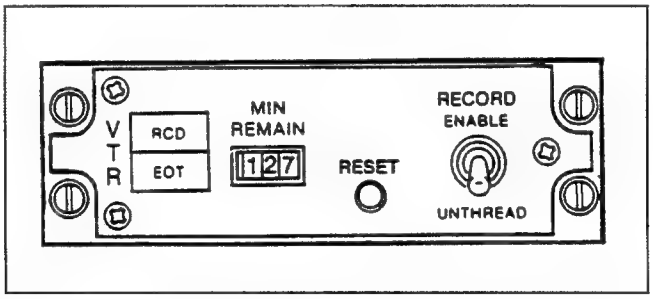


Figure A.3-7. Video Tape Recorder Control Panel Schematic.

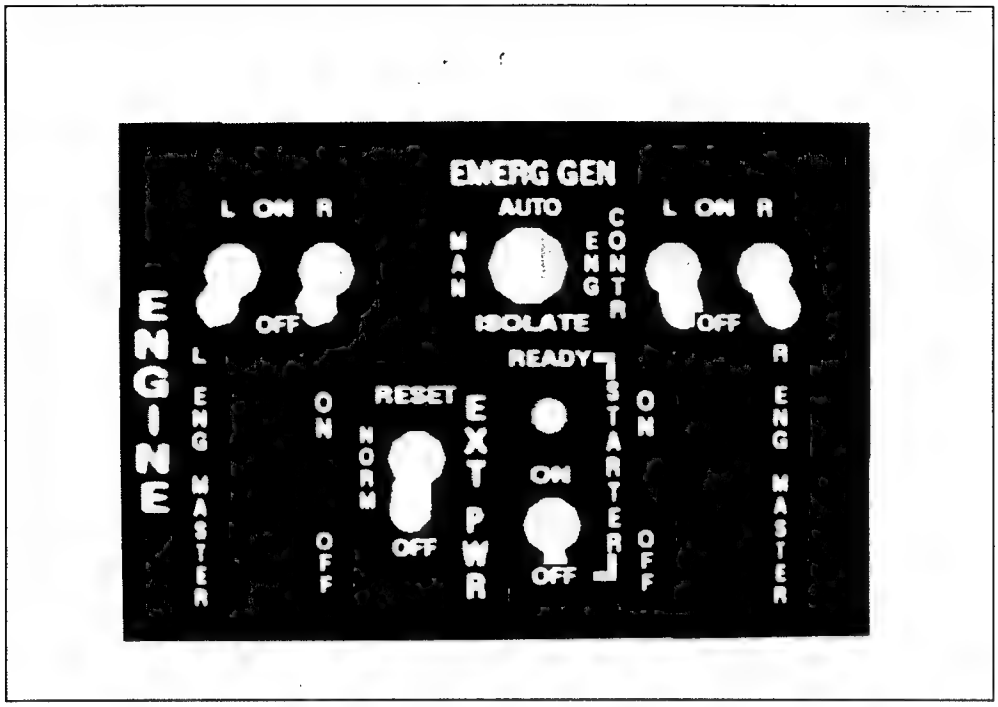


Figure A.3-9. Engine Control Panel Model.

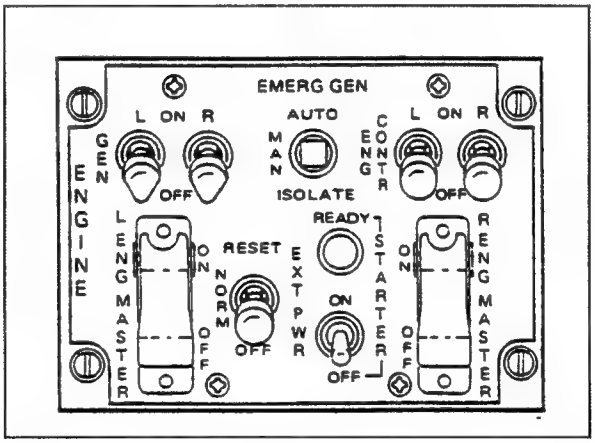


Figure A.3-10. Engine Control Panel Schematic.

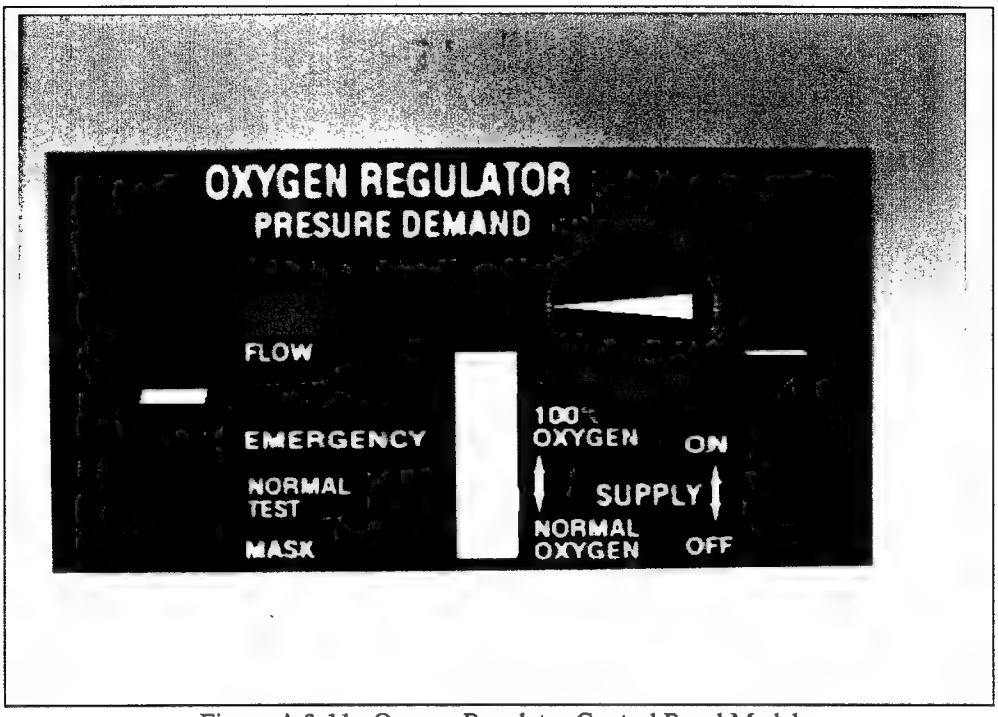


Figure A.3-11. Oxygen Regulator Control Panel Model.

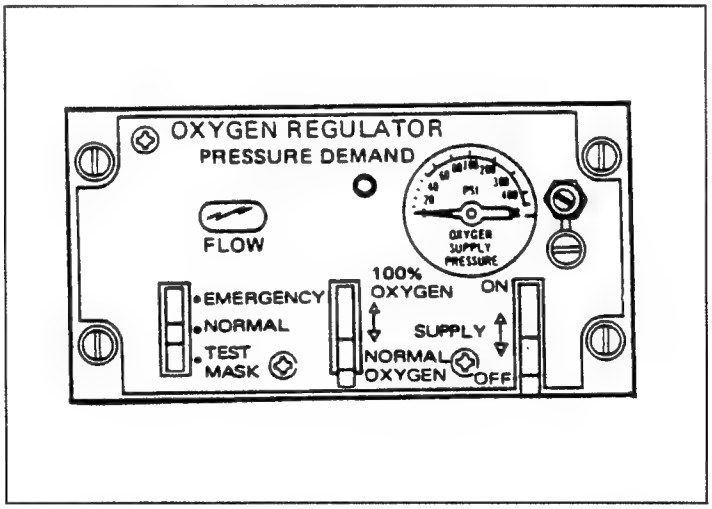


Figure A.3-12. Oxygen Regulator Control Schematic.

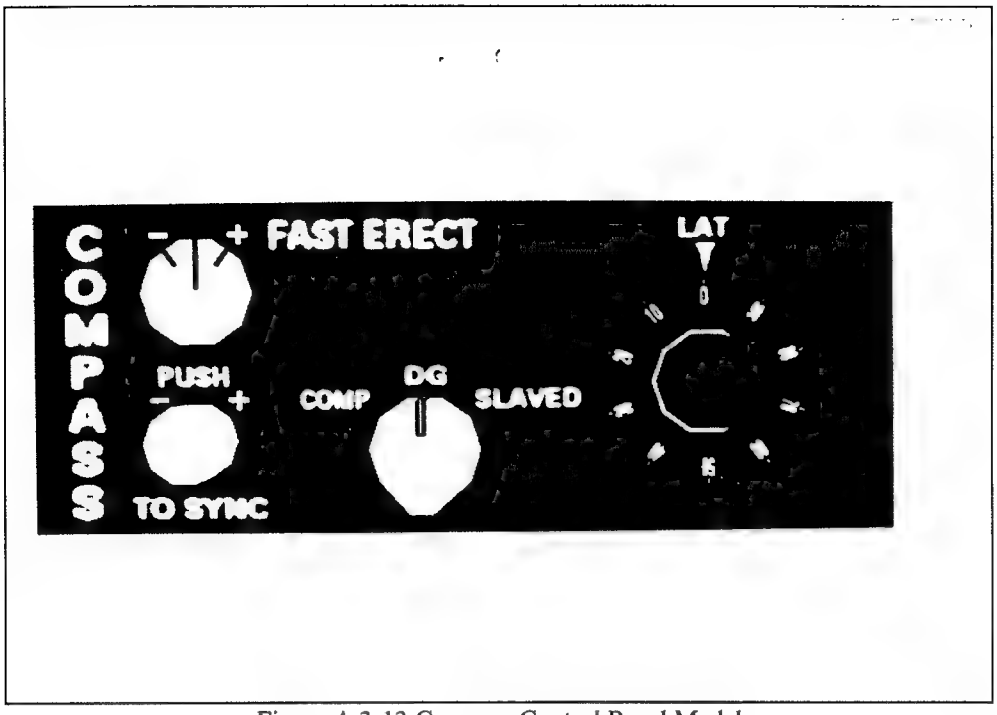


Figure A.3-13 Compass Control Panel Model

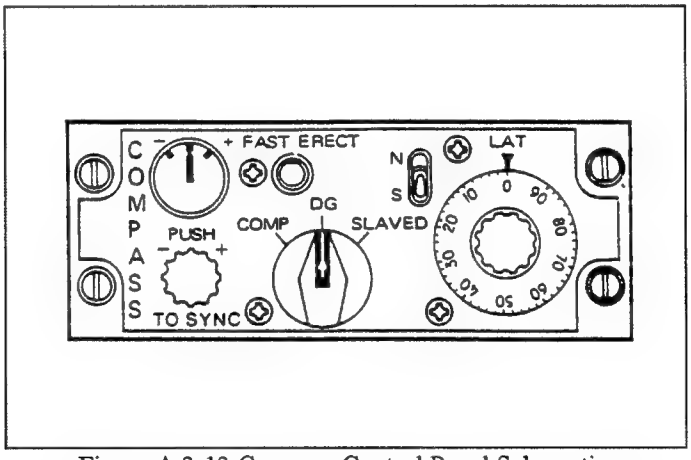


Figure A.3-13 Compass Control Panel Schematic.

Appendix B

B.1 Introduction

Appendix B presents the series of steps I used to: 1) place a model in the VC's environment, 2) place the dynamic switch model, and 3) position the mouse-panel and button. This appendix offers an example illustrating the three step process for co-locating a mouse-panel button with the nuclear control switch. Section B.2 is a brief description of the code used to render the static and dynamic parts which make up the left console. The placement of mouse-panels and buttons are described in sections B.3 and B.4 respectively. Section B.5 shows the header files "leftconsole.h," "left_panel.h," and "nuc_panel.h".

B.2 Displaying the Static and Dynamic Models

The nuclear control panel is located just forward of the throttle on the left console. This panel, like the others on each console, is a static display. Rather than position each static panel separately, the static parts of left console are placed as one model (leftconsole) into the simulation.

```
// Class: LEFTCONSOLE_CLASS
//
// Purpose: Call back function for drawing left console readouts.
//
// Author:  Milton Diaz
//
// Written in AT&T C++.
//
// Released into the public domain.
//////////////////////////////////////////////////////////////////////

#include "sim.h"
#include "sim_models.h"
#include "leftconsole.h"

void LEFTCONSOLE_CLASS::init()
 {
  int found;
  pfMatrix m,n,scale,orient;

  // Create oxygen generator as an object in the simulation
  left_console = new Flt_Model();
  left_console->readmodel(CONSOLES_FILE, CONSOLES_ID, found);
```

The scale matrix and translation vectors do not change the model because the model is positioned correctly. The rotational matrix applied to the model, identical to the matrix applied to the VC's F-15E, correctly orients the console in the virtual environment.

```
  // Add it to the scene in Y out the nose, Z up coords
  pfMakeScaleMat(scale,1.0f, 1.0f, 1.0f);                    // scaling matrix
  // positive x = out the planes right wing tip
```

```
pfMakeTransMat(m,  -0.0f,  0.0f,  0.0f);                    // translation vector
pfMakeEulerMat(orient,-90.0f, 0.0f, 180.0f);                // Rotation matrix
pfMultMat(n,orient,scale);
pfMultMat(m,n,m);
scale_location = pfNewSCS(m);
pfAddChild(Ac->Model->root, scale_location);
pfAddChild(scale_location,left_console->root);
```

The nuclear control panel's dynamic parts are the red switch cover and the switch beneath the cover. Like the console, the switches are also multiplied by a unit value scaling matrix. The rotation matrix is identical to the left console's rotation in order to align the switch with the VC's F-15E model. However, unlike the static models, the dynamic parts are placed at the origin within the MultiGen modeling environment rather than at the desired location within the F-15E model. By placing dynamic models at the origin, finding the rotation becomes trivial.

To find the needed dynamic model translation, location of the static switch's origin is found using MultiGen. The translation values are equal to the model xyz coordinates values displayed by MultiGen. For example, the red switch cover values used in the "pfMakeTransMat (m, -0.239f, 5.586f, 1.0f);" are derived from the corresponding MultiGen values: X= 5.586, Y= -0.239, and Z= -1.0.

```
//-------------------------------------------------------//
// The Nuclear Control panel has one covered switch     //
// This code establishes the starting positions of the  //
// switches.
//
  // RED SWITCH COVER
  redswitch_cover = new Flt_Model();
  redswitch_cover->readmodel(REDSWITCHCOVER_FILE, REDSWITCHCOVER_ID, found);

  // Add it to the scene in Y out the nose, Z up coords
  pfMakeScaleMat(scale,1.0f, 1.0f, 1.0f);                   // Scale Matrix
  // positive x = out the planes right wing tip
  pfMakeTransMat(m, -0.239f, 5.586f,  1.0f);                // Translation vector
  pfMakeEulerMat(orient,-90.0f, 0.0f, 180.0f);              //  Rotation Matrix
  pfMultMat(n,orient,scale);
  pfMultMat(m,n,m);
  scale_location = pfNewSCS(m);
  redswitch_cover->RotDCS = pfNewDCS();
  pfAddChild(Ac->Model->root, scale_location);
  pfAddChild(scale_location, redswitch_cover->RotDCS);
  pfAddChild(redswitch_cover->RotDCS, redswitch_cover->root);

  // NUC SWITCH ----------------------------------------------//
  nucswitch = new Flt_Model();
  nucswitch->readmodel(STANDARDSWITCH_FILE, STANDARDSWITCH_ID, found);
  // Add it to the scene in Y out the nose, Z up coords
  pfMakeScaleMat(scale, 1.0f, 1.0f, 1.0f);                  // scaling matrix
  // positive x = out the planes right wing tip
  pfMakeTransMat(m, -0.217f, 5.590f,  1.001f);              // tranlation vector
  pfMakeEulerMat(orient,-90.0f, -20.0f, 180.0f);            // orientation matrix
```

```
pfMultMat(n,orient,scale);
pfMultMat(m,n,m);
scale_location = pfNewSCS(m);
nucswitch->RotDCS = pfNewDCS();
pfAddChild(Ac->Model->root, scale_location);
pfAddChild(scale_location, nucswitch->RotDCS);
pfAddChild(nucswitch->RotDCS, nucswitch->root);
}

// Function: draw_leftconsole
////////////////////////////////////////////////////////////////////
void LEFTCONSOLE_CLASS::draw_leftconsole()
 {
 }
```

Displaying changes in the status of dynamic models is accomplished in the propagate function. The model is adjusted by toggling through the set of possible positions until the desired position is reached. For example, the nuclear control panel has two moving parts controlled by one button. When the button is activated, the "Globals->Nuc_Arm_Switch" condition becomes true and the two moving parts are redrawn based on the previous condition.

```
void LEFTCONSOLE_CLASS::propagate()
 {
 if (Globals->Nuc_Arm_Switch)
  {
  static int Nuc_Arm_Switch_Angle;
  switch (Nuc_Arm_Switch_Angle)
   {
   case  0:                                  // If the red switch cover is closed, open it!
    Nuc_Arm_Switch_Angle = 1  ;        //   Set condition = condition 1
                          // condition 1 means Red Switch Open and to Toggle switch  is safed
    pfDCSRot(redswitch_cover->RotDCS, 0.0f, -60.0, 0.0f);
    break;
   case 1:                                   // Condition 1 then set toggle to jettison enable
    Nuc_Arm_Switch_Angle = 2 ;         //     Set condition = Condition 2
    pfDCSRot(nucswitch->RotDCS, 0.0f, -40.0, 0.0f);
    break;
   case 2:                                   // Condition 2 then  safe & close switches
    Nuc_Arm_Switch_Angle = 0 ;         //     Set condition to Zero condition
    pfDCSRot(redswitch_cover->RotDCS, 0.0f, 0.0, 0.0f);
    pfDCSRot(nucswitch->RotDCS, 0.0f, 0.0, 0.0f);
    break;
   default:
    Nuc_Arm_Switch_Angle =  0 ;        // precaution to reset condition if values is
   }                                         //   out of range.
  Globals->Nuc_Arm_Switch   = FALSE;
  }
 }
```

B.3 Positioning Panels

The left_panel.cc places a mouse-panel over the left console model in the virtual environment. The mouse panel defines the area upon which the mouse is confined. The left console's four corners are passed to the mouse cursor routines by the "Get_Point_x" functions shown below. The console's corner coordinate values were found with the MultiGen modeling tool.

```
//=============================================================================
//
// File:        left_panel.cc
//
// Description: This is the class definition for the left  panel
//
// Author:      Milton Diaz
// Date:         June 94
//
// Adapted from  the AFIT Information Pod Code by Vanderburgh, Kestermann, & Rohrer
// Date:        May 94
//
//=============================================================================
#include "left_panel.h"
#include <iostream.h>

void Left_Panel_Type::Update_Children(pfSeg *Segment, int Mouse_Button_Status, pfVec3
X_Finger_Marker)
 {
  //------------------------ Update subpanels -----------------------------
  nuc_panel.Update_Base   (Segment, Mouse_Button_Status, X_Finger_Marker);
 }


void Left_Panel_Type::Get_Point_1 (pfVec3& Point)
 {
 PFSET_VEC3 (Point, 0.0f, 0.579f, 0.0f);
 };


void Left_Panel_Type::Get_Point_2 (pfVec3& Point)
 {
 PFSET_VEC3 (Point, 0.193f, 0.579f, 0.0f);
 };


void Left_Panel_Type::Get_Point_3 (pfVec3& Point)
 {
 PFSET_VEC3 (Point, 0.193f, 0.0f, 0.0f);
 };
```

```
void Left_Panel_Type::Get_Point_4 (pfVec3& Point)
 {
 PFSET_VEC3 (Point, 0.0f, 0.0f, 0.0f);
 };



void Left_Panel_Type::Register_Callbacks ()
 {
   pfNodeTravFuncs (Opaque_Geometry,
              PFTRAV_DRAW,
              NULL,
              &Left_Panel_Draw_Stuff);

   pfNodeTravData  (Opaque_Geometry,
              PFTRAV_DRAW,
              this);
 };
```

The sub-panel which contains the nuclear panel's switches and the "X" shaped cursor are defined in the initialize_children routine. The values passed in "nuc_panel.Init_Base (0.096f, 0.489f, 0.0f, *Get_Matrix(), 0.01f, 0.01f, 0.01f); " are the xyz-coordinate values, the parent panel's coordinate matrix, and the xyz-scaling factor. The xyz-coordinate values are also found using MultiGen. Also, the coordinate values are found relative to the coordinates of "point_1." The Additional sub-panels would be initialized in a similar manner.

```
void  Left_Panel_Type::Initialize_Children ()
 {

   //------------------------ Initialize sub-panels -------------------------
   nuc_panel.Init_Base   (0.096f,  0.489f,  0.0f, *Get_Matrix(), 0.01f, 0.01f, 0.01f);

   //--------------------- Set up cursor feedback flags -------------------
   Set_X_Finger_Marker(1);  //<- true
 };


//-----------------------------------------------------------------------------
// Draw_Stuff
//
// Try drawing stuff on the panel.  Because this is a function the member
// information is passed in as Data and then typecast to the class
// that it came from.  You can now access information in the class
// by dereferencing the variable "This" (note the capital "T")
//-----------------------------------------------------------------------------
long Left_Panel_Draw_Stuff (pfTraverser* T, void* Data)
{
  pfPushState ();
  pfBasicState ();
  Globals->Z_Offset_Flag = FALSE;
```

```
    //---------------------- Convert to member function ----------------------
    Left_Panel_Type *This = (Left_Panel_Type*)Data;

    //------------------------ Call base draw routine ------------------------
    // *** !!! Please always call this routine in the panel callback
    This->Draw_Base();
```

The "This->nuc_panel.Draw_Base();" function takes care of putting mouse activated buttons into the simulation. This will be shown in the next section.

```
    //----------------------------- Draw children ----------------------------
    This->nuc_panel.Draw_Base();

    pfPopState ();

    return PFTRAV_CONT;
}
```

B.4 Positioning Sub-Panels and Buttons

The Nuc_Panel_Type is a sub-panel type. This code initializes the positions of all the buttons on the sub-panel. The "button init2" call specifies the rectangular size of the buttons.

```
//=============================================================================
// File:       nuc_panel.cc
//
// Description: This is the class definition for the nuc_panel sub-panel
//
// Authored by:  Milton Diaz
// Date:             June 1994
// Adapted from code developed by:     Vanderburgh, Kestermann, & Rohrer
// Date:       May 94
//
//=============================================================================
#include "nuc_panel.h"
#include "GraphText.h"          // used to keep text in perspective size
#include "labfont.h"            // used to label button
#include "common_pod_colors.h"  // used to pass colors to button

void Nuc_Panel_Type::Initialize_Children ()
{
 Globals->Nuc_Arm_Switch = FALSE;
 //---------------------- Set up buttons ----------------------------
 // Input parameters to class Button_Type.Init:
 // ( button size, subpanel matrix, postion x,y,z, "on"-color, "off-color"-color,
                                                          "on"-text, "off"-text )

 //---------------------- Position the CONTROL button ---------------------
 Nuclear_Consent_Button.Init2 (3.0f, 1.0, Current_Matrix,
   (short)Button_Type_2::NONE, 3.8f, 1.3f, 0.0f,
    Red_Color, Medium_Gray_Color, "   ", "   " );
}
```

The sub-panel checks for the following conditions: 1) Mouse_Left_Button_On is true (the left button was pressed), 2) Push_Last_Frame is false (prevents multiple toggle signals from one press), and 3) "X_On_Button2(Pointer_Finger, Nuclear_Consent_Button, X_Finger_Marker)" is true (meaning the mouse cursor is on a button). When all these conditions are true, an intersection has occurred. The global flags are set to "TRUE" in the intersects code section below. Other sections of code will use the globals set here to generate the proper response. For example, the left console code in section B.2 uses the "Globals->Nuc_Arm_Switch" flag to toggle the cycle through the possible switch positions.

```
void Nuc_Panel_Type::Update_Children (pfSeg* Pointer_Finger,
                    int Mouse_Left_Button_On, pfVec3 X_Finger_Marker)
{
  static int Push_Last_Frame = 0;  //<-- used to toggle button in place

  //--- handle button actions
  if ( !Mouse_Left_Button_On)
    Push_Last_Frame = 0;

 //---------------------- Check to see if intersection -----------------
 if ((!Push_Last_Frame) &&
      Mouse_Left_Button_On &&
      X_On_Button2(Pointer_Finger, Nuclear_Consent_Button, X_Finger_Marker))
  {
    // return the pod to its initial starting position from when the program began
    Push_Last_Frame = 12;
    Nuclear_Consent_Button_State = 1;
    Set_ARM_Switch (Globals->Nuc_Arm_Switch);
      // Set Additional Globals Here!
  }
 else
    Nuclear_Consent_Button_State = 0;
}
```

The VC's active button set is not normally drawn. However, the ability to draw buttons remains for debugging and reprogramming purposes. When adding a new button, a programmer will need to use this feature to verify the buttons are where he/she placed them.
Thus, each new button added should also be inserted here.

```
void Nuc_Panel_Type::Draw_Children()
{
 if (Globals->View_Buttons)   // if we need to draw buttons then continue
  {
  pushmatrix();

  //------------------------ Draw the button -----------------------
  Nuclear_Consent_Button.Draw2 (Nuclear_Consent_Button_State);
  popmatrix();
  } //endif
}
```

For each intersection tested in the above routine, the global variable corresponding to the selected button should be set. These variables may be set above or as a procedure. The following example is a procedural toggle. For panels with more than one switch, the procedural approach yields a cleaner code.

```
// ------------------------------------------------------------------------//
void Nuc_Panel_Type::Set_ARM_Switch (boolean Flag)
{
 Globals->Nuc_Arm_Switch = !Flag;
}
```

B.5 Header Files

The header files for the code discussed in sections B.2-4 is presented below. The mouse panels technique is based on the AFIT Information Pod developed at AFIT (Kestermann; Stytz and others).

The leftconsole header file.

```
#include "sim.h"
#ifndef __LEFTCONSOLE_CLASS
#define __LEFTCONSOLE_CLASS
#include <ulocks.h>
#include "airplane_test.h"
#include "global_declarations_test.h"
#include "flt_model.h"
class Airplane;
struct leftconsole_struct
 {
 };

class LEFTCONSOLE_CLASS
 {
 private:
   pfSCS*    scale_location;
   Flt_Model* left_console;
   Flt_Model* redswitch_cover;   // NUC CONTROL PANEL
   Flt_Model* nucswitch;        // NUC CONTROL PANEL
//   Flt_Model* emer_generator;
//   Flt_Model* fast_erect;
//   Flt_Model* vtr_recorder;
//   Flt_Model* edit_edition;


 public:
   Airplane* Ac;

   void init();
   void draw_leftconsole();
   void propagate();
 };
#endif
```

The left_panel header file.

```
//=============================================================================
//
// File:       left_panel.h
//
// Description: This is the class definition for the left  panel
//
//
//=============================================================================

#ifndef __LEFT_PANEL_TYPE_H
#define __LEFT_PANEL_TYPE_H

#include "panel_type_test.h"
#include "button_type.h"
#include "nuc_panel_test.h"
#include "global_declarations_test.h"
#include "mRS232port.h"

//----------
// Callbacks:
//----------
extern long Left_Panel_Draw_Stuff  (pfTraverser*, void*);

class Left_Panel_Type : public Panel_Type
{
 //-------------------------- Sub-panels --------------------------
 Nuc_Panel_Type nuc_panel;
 //-------------------------- Other stuff --------------------------
 friend long Left_Panel_Draw_Stuff  (pfTraverser*, void*);
 void Get_Point_1 (pfVec3& Point);
 void Get_Point_3 (pfVec3& Point);
 void Get_Point_4 (pfVec3& Point);
 void Register_Callbacks ();
 void Initialize_Children ();

public:
 void Update_Children(pfSeg *Segment, int Mouse_Button_Status, pfVec3 X_Finger_Marker);
 void Get_Point_2 (pfVec3& Point); //put this down here to get panel extents
};
#endif
```

The nuc_panel header file.

```
//=============================================================================
//
// File:       nuc_panel.h
//
// Description: This is the class definition for the Nuc_panel sub-panel that
//
//
//=============================================================================
#ifndef _nuc_panel_H
#define _nuc_panel_H

#include "button_type.h"
#include "button_type_2.h"
#include "sub_panel_type.h"
#include "global_declarations_test.h"

class Nuc_Panel_Type : public Sub_Panel_Type
{
 private:
 //--------------------------- Button types ---------------------------
   Button_Type_2 Nuclear_Consent_Button;
 //--------------------------- Button state ---------------------------
   int Nuclear_Consent_Button_State;   // 1 = yes, 0 = no

   void Set_ARM_Switch (boolean Flag);

 public:
 //------------------ Mandatory Virtual Stuff -------------------------
   void Initialize_Children ();
   void Update_Children (pfSeg* Pointer_Finger,
                         int Mouse_Left_Button_On, pfVec3 X_Finger_Marker);
   void Draw_Children();
 };
#endif
```

Appendix C

C.1 Running the Virtual Cockpit

To start the VC without the HMD, follow these steps:

1) enter the directory with the executable file named "plane"

2) ensure the third line of fastrak.dat is "0".

3) ensure the HOTAS is connected to the system in port 1

4) at the keyboard, enter "plane<cr>"

5) you must initialize the HOTAS if the HOTAS is being used for the first time or another HOTAS has been previously initialized

6) the program will take approximately 30 seconds to start

To start the VC with the HMD, follow these steps:

1) enter the directory with the executable file named "plane"

2) "1" must be the value on the third line of fastrak.dat

3) ensure the HOTAS is connected to the system in port 1

4) ensure the Fastrak is connected to the system in port 2 and the system has started The VC takes care of running and shutting down the application

5) at the keyboard, enter "plane<cr>"

6) you must initialize the HOTAS if the HOTAS is being used for the first time or another HOTAS has been previously initialize.

7) to initialize the Fastrak, wear the HMD in the pilot's position and enter <cr> when prompted

8) the program will take approximately 30 seconds to start

The eye point in HMD and CRT modes may be adjusted by the U, D, F, B, L, and R keys. These keys are used to set the pilot in a comfortable position in the VC. The position is adjusted permanently; although the movements may always be reversed, they can not be reset during the simulation.

U	move eye point up 0.5 feet (in the virtual environment)
D	move eye point down 0.5 feet
F	move eye point forward 0.5 feet
B	move eye point back 0.5 feet
L	move eye point left 0.5 feet
R	move eye point right 0.5 feet

The arrow keys and number pad only move the eye point in CRT keys are used to look around when in the CRT mode.

↑	move eye point up 1.0 foot
↓	move eye point down 1.0 foot
→	move eye point right 1.0 foot
←	move eye point left 1.0 foot
1	roll view counterclockwise
2	pitch view down
3	roll view clockwise
4	turn view heading left
5	RESET VIEW
6	turn view heading right
7	not used
8	pitch view up
9	not used
ENTER	move view point forward 1 foot
+	move view point backwards 1 foot

Other commands used at the keyboard.

Q	Display mouse-panel buttons
P	Pause simulation
F1	display performer statistics
F2	toggle HMD tracking on
F3	Toggle CRT on, HMD tracking off
F8	reset simulation
ESC	QUIT
Del	turns on Radar; do not turn radar on when the net is empty This key is needed for HOTAS which are not have fully functional.

C.2 Global Declarations

The Global declarations declared in the 1994 VC are listed. These variables are used in addition to the globals declared in previous work.

```
//////////////////////////////////////////////////////////////////////
// global_types.h
// Defines all type defined structures and constants used throughout
// the Virtual Cockpit system.
// History:
//        1 May 93          Initial Development
//
//////////////////////////////////////////////////////////////////////
#ifndef _GLOBAL_TYPES
#define _GLOBAL_TYPES

#include "pf.h"
#include "globals.h"
//------------------ New 1994 Net stuff ------------------
#include "DIS_v2_cockpit_obj_mgr.h"
#include "vc_net_manager.h"
#include "sim_entity_mgr.h"
#include "model_mgr.h"

//////////////////////////////////////////////////////////////////////
//  STRUCTURE AND ENUM TYPES
//////////////////////////////////////////////////////////////////////

//-------------------------------------
// Global stuff: (put in shared memory)
//-------------------------------------
typedef struct
{
   // LEFT HAND SIDE OF COCKPIT
   boolean Nuc_Arm_Switch;                  // NUCLEAR CONTROL PANEL
```

The global variables are set when the button corresponding to the variable is set. The Oxygen Control Panel switches are the only working switches on the right hand side of the cockpit. The Emergency_Oxygen_Switch_Angle and Percent_Oxygen_Switch_Angle variables are used to keep track of the switches position.

```
   // RIGHT HAND SIDE OF CONTROL PANEL
   boolean Emergency_Oxygen_Switch;        // OXYGEN REGULATOR PANEL
   int     Emergency_Oxygen_Switch_Angle;
   boolean On_Off_Oxygen_Switch;           // OXYGEN REGULATOR PANEL
   boolean Percent_Oxygen_Switch;          // OXYGEN REGULATOR PANEL
   int     Percent_Oxygen_Switch_Angle;
```

The buttons co-located with each of the active switches and knobs on the VC front instrument panel is listed here according by panel. Not all buttons have been activated. For example, buttons on the navigation panel's key pad remain to be activated.

```
// INSTRUMENT PANEL
boolean Nav_Sym_Brt_Knob,              // NAV PANEL
    Nav_LCD_Brt_Knob,
    Nav_Vidio_Brt_Knob,
    Nav_Vidio_Contrast_Knob,
    Nav_Radio13_Vol_Knob,
    Nav_Radio24_Vol_Knob,
    Armt_Safe_Switch,              // ARMAMENT CONTROL PANEL
    Armt_Select_Knob,
    AMAD_Discharge_Switch,         // AMAD FIRE-WARNING PANEL
    AMAD_Warning_Lt_Switch,
    AMAD_LEngine_Warning_Lt_Switch,
    AMAD_REngine_Warning_Lt_Switch,
    Fuel_Quantity_Knob,            // FUEL QUANTITY PANEL
    Gear_Knob,                     // LANDING GEAR CONTROL PANEL
    Gear_Nose_Lt,
    Gear_Left_Lt,
    Gear_Right_Lt,
    Warning_Master_Caution_Switch, // WARNING LIGHTS
    Warning_Spare_1_Switch,
    Warning_Spare_2_Switch,
    Warning_Spare_3_Switch,
    Warning_Spare_4_Switch,
    Warning_Emis_Lmt_Switch,
    Warning_Ai_Sam_Switch,
    Warning_Low_Alt_Switch,
    Warning_Obst_Switch,
    Warning_Tf_Fail_Switch,
    Warning_Can_Unkld_Switch,
    Warning_Laser_Armed_Switch;


// GENERAL AirCraft CONDITION
boolean Afterburner,                   // If the aircraft is in AB then the fuel
                                       consumption routine will need to know. This flag
                                       raises the issue.
    Reset_AcftState_Position,          Set when reset command is given
    View_Buttons,                      // AIRCRAFT VIEW FLAGS
    Update_View_Switch,                // Switch Turns on the radar update displays
    radar_cursor_toggle,               // Toggle radar cursor movement rules
    Z_Offset_Flag;
pfMatrix translate,scale,orient,multimatrix;
```

These variables are used by the network, model, and entity managers.

```
    // NETWORK STUFF
    VC_Net_Manager    NetMan;
    Sim_Entity_Mgr    SimMan;
    Sim_Entity_Mgr*   Netobj;
    Model_Manager     Net_Mod_Man;
    DIS_v2_cockpit_object_manager  SendMan;
    DIS_v2_cockpit_object_manager* Net;

} Shared;

#endif
```

Bibliography

Aeronautical Systems Division. *F-15E Human Engineering Design Approach Document-Operator (DID No. DI-H-7056/T).* Revision D (Final Submission); Contract F33657-84-C-2228. Ohio: Wright-Patterson Air Force Base, 1 May 1988.

Astheimer, Peter, Wolfgang Felger, and Stefan Muller. "Virtual Design: A Generic VR System For Industrial Applications," *Computer & Graphics; An International Journal* 17: 671-679, 1993.

Aukstakalnis, Steve and David Blatner. *Silicon Mirage: The Art and Science of Virtual Reality.* Peachpit Press, Berkeley, CA: 1992.

Bess, Rick D. "Image Generation Implications for Networked Tactical Training Systems," *1993 IEEE Annual Virtual Reality International Symposium.* 308-317. NJ: IEEE, 1993.

Bryson, Steve. "Virtual Reality In Scientific Visualization," *Computer & Graphics; An International Journal* 17: 679-685, 1993.

Bryson, Steve and Creon Levit. "The Virtual Windtunnel: An Environment for the Exploration of the Three-Dimensional Computer Generated Flowfields," *IMAGE VI Conference*: 137-139, Arizona 1992.

Butterworth, Jeff, Andrew Davidson, Stephen Hench, and T. Marc Olano. "3DM: A Three Dimensional Modeler Using a Head-Mounted Display," *Proceedings 1992 Symposium on Interactive 3D Graphics*: 135-138, Massachusetts, 29 March-1 April 1992.

Clausewitz, Carl Von. *On War.* Princeton, NJ: Princeton University Press, 1989.

Cooke, Joseph M., Michael J. Zyda, David R. Pratt, and Robert B. McGhee. "NPSNET: Flight Simulation Dynamic Modeling Using Quaterions," *Presence* 4. 404-421, 1992.

"DIS Mixes Real, Virtual," *Aviation Week & Space Technology.* 73: May 9, 1994.

DeHaemer, Michael J., and Michael J. Zyda. "Simplification of Objects Rendered by Polygonal Approximations," *Computers & Graphics, 13*: 175-184, 1991.

Dyer, Gwynne. *War.* New York, NY: Crown Publishers, 1985.

Earle, Steph. Chief of Training and Evaluation, 448th Missile Squadron, Grand Forks AFB, ND. Personal interview. 20 September 1994.

Erichsen, Matthew Nick. *Weapon System Sensor Integration for a DIS-Compatible Virtual Cockpit.* MS Thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/93-07, December 1993.

Figueiredo, Mauro, Klaus Bohm, and Jose Teixeira. "Advanced Interaction Techniques in Virtual Environments," *Computer & Graphics; An International Journal* 17: 655-662, 1993.

Furness, Thomas A. III and Dean F. Kocian. "Putting Humans Into Virtual Space," Armstrong Aerospace Medical Research Laboratory, WPAFB, OH.

Gerhard, William Edward Jr. *Weapon System Integration for the AFIT Virtual Cockpit.* MS Thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/93-10, December 1993.

Hazer, Kaleem Jr. and Ringler, Daniel L. *Applicability of Design-To-Cost to Simulator Acquisition.* MS Thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, SLSR 36-76A, June 1976.

Institute for Simulation and Training, 12424 Research Parkway, Suite 300, Orlando FL 32826. *Proposed IEEE Standard Draft Standard for Information Technology - Protocols for Distributed Interactive Simulation Applications Version 2.0 Second Draft,* March 1993. Contract Number N61339-911-C-0091.

Kestermann, Jim B. *Immersing the User in a Virtual Environment: The AFIT Information Pod Design and Implementation.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/94D-13, December 1994.

Kriss, Jordon and Paul Kubiak. Air Force Human Factors Lab/Crew Station Evaluation Facility, Wright-Patterson Air Force Base. Personal interview. 10 March 1994.

Lewis, Christopher. President, n-Vision. Personal interview. 17 September 1994.

Ling, Daniel T. *Beyond Visualization - Virtual Worlds for Data Understanding*. Research Report RC 15479 (#68850). IBM Research Division, T.J. Watson Research Center, NY: IBM, 2/9/90.

Markman, Steven R. "Capabilities of Airborne and Ground Based Flight Simulation," in *Flight Simulation/Simulators.* Aerospace Technology Conference & Exposition, Technical Session. 35-42. PA: Society of Automotive Engineers, Inc., October, 1985.

McCarty, W. Dean. *Rendering the Out-the-Window View for the AFIT Virtual Cockpit.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/93M-04, March 1993.

McLendon, Patricia. IRIS Performer Programming Guide. Silicon Graphics Inc., Mountain View, California, 1992.

Rohrer, Jimmie J. *Design and Implementation of Tools to Increase user Control and Knowledge Elicitation in a Virtual Battlespace.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/94D-20, December 1994.

Rolfe, J.M. and K.J. Staples, editors. *Flight Simulation.* New York, NY: Cambridge University Press, 1986.

Ross, Milton C. and Gerald L. Yarger. *A Parametric Costing Model For Simulator Acquisition.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, SLSR 22-76B, September 1976.

Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall 1991.

Schwarzkopf, H. Norman and Peter Petre. *It Doesn't Take A Hero.* New York, NY: Bantam Books, 1992.

Software Systems. "MultiGen Modeler's Guide: Revision 14," San Jose, CA: March 1994.

Roe, David B. and Jay G. Wilson. "Whither Speech Recognition: The Next 25 Years," *IEEE Communications Magazine*: 54-62: November 1993.

Stytz, Martin R. Professor, School of Engineering (EN), Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH. Personal interview. 21 November 1994.

Stytz, Martin R., Steven S. Sheasby, Keith Shomper, Jim Kestermann, J.J. Rohrer, and John C. Vanderburgh. "Software Architecture and User Interfaces for Distributed Virtual Environments," Submitted for publication. 1994.

Sutherland, Ivan. "The Ultimate Display," *Proceedings of the IPIP Congress* 2. 506-508. 1965.

Switzer, John C. *A Synthetic Environment Flight Simulator The AFIT Virtual Cockpit.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/92D-17, December 1992.

Taylor, S. Craig. *Wooden Ships and Iron Men: Naval Warfare During the Age of Sail.* Baltimore, Maryland: Avalon Hill, 1975.

Vanderburgh, John C. *Space Modeler: An Expanded, Distributed, Virtual, Environment for Space Visualization.* MS thesis, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base, OH, AFIT/GCS/ENG/94D-23, December 1994.

Venolia, Dan. "Facile 3D Direct Manipulation," in Human Factors in Computing Systems, *INTERCHI '93 Conference Proceedings.* ACM/SIGCHI Amsterdam, Netherlands. 31-36: 1993.

Zeltzer, David and Steven Drucker. "A Virtual Environment System for Mission Planning," *Proceeding of the 1992 IMAGE VI Conference*. 125-133. Scottsdale, Arizona: July 1992.

Zyda, Michael J., David R. Pratt, John S. Falby, Chuck Lombardo, and Kristen M. Kelleher. "The Software Required for the Computer Generation of Virtual Environments," *Presence: Volume 2. Number 2*. 130:1993.

Vita

Capt Milton E. Diaz was born in San Juan, Puerto Rico, on 26 January 1960, the youngest of four children. After graduating from Piedmont Hills High School, in San Jose, California, he attended University of California Davis and earned a Bachelor of Science degree in Electrical and Computer Engineering in 1983. On 28 March 1994, he was commissioned an officer in the Air Force and served as Fire Control Project Engineer at the Avionics Laboratory, Wright-Patterson AFB, Ohio. He was next assigned to the 448th Strategic Missile Squadron in Grand Forks AFB, North Dakota from 1988 to 1992. While at the 448th , he served as Senior Flight Commander and Chief of Trainer Maintenance until he was selected to pursue graduate studies at the Air Force Institute of Technology, Dayton. Capt Diaz was graduated from AFIT with a Master of Science degree in Computer Science in December 1994. Milton & Florence M. Diaz are currently stationed at Los Angeles Air Force Base.

Permanent address: 1121 Caballo Court
San Jose, California 95132

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
THE PHOTO-REALISTIC AFIT VIRTUAL COCKPIT

5. FUNDING NUMBERS

6. AUTHOR(S)
Milt E. Diaz, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ARPA/ASTO
3701 North Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Air Force Institute of Technology (AFIT) has pursued research in virtual environments since 1988. This research expands the current capabilities of the AFIT Virtual Cockpit (VC) by increasing the realism of the cockpit environment and improving the pilot's command interface. Realism is improved creating console elements from texture maps and polygonal models; these elements include working dials, switches and circuit breakers. The pilot command interface is improved in part by adapting the AFIT Information Pod using a two-dimensional mouse input to the virtual three-dimensional environment. This immersive virtual environment is also improved by modifications to the Head Mounted Display (HMD) reducing jitter and allowing the simulation pilot to adjust his/her position within the cockpit.

14. SUBJECT TERMS	15. NUMBER OF PAGES
synthetic environments, virtual reality, flight simulator, distributed interactive simulation, computer graphics	112
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

* U.S. GPO 1990-0-273-271